객체지향 및 고전 쏘프트웨어공학

김일성종합대학출판사
주체91

객체지향 및 고전 쏘프트웨어공학

김일성종합대학출판사

차　례

제3장. 쏘프트웨어생명주기모형

제4장. 개발팀

제5장. 거래되고 있는 도구

제6장. 시 험

제7장. 모듈로부터 객체에로

제8장. 재리용성, 이식성, 호상조작성

제9장. 계획작성과 타산

제2편. 쏘프트웨어생명주기의 단계

제10장. 요구사항확정단계

제11장. 명세작성단계

제12장. 객체지향분석단계

제13장. 설계단계

제14장. 실현단계

제15장. 실현 및 통합단계

제16장. 유지정비단계

머 리 글

이 책의 4판은 두개의 판본으로 출판되였다. 즉 한 판본에서는 C++, 다른 판본에는 Java로 코드실례를 제시하였다. 그러나 쏘프트웨어공학은 본질적으로 언어에 의존하지 않으므로 이 책에서는 상대적으로 코드실례를 적게 제시하였다. 이번 출판에서는 언어에 의존하는 상세한 내용들은 피함으로써 제시된 코드실례들이 C++와 Java사용자들에게 모두 명백하도록 하기 위하여 노력하였다. 실례로 C++에서는 출력명령으로 cout를, Java에서는 출력명령으로 system.out.println을 리용하지만 여기서는 대신에 의사명령 print를 리용하였다(새로운 실례연구가 하나의 례외로 되는데 여기서는 완전한 실현세부를 C++와 Java로 제시한다.). 그러므로 5판은 4판의 두 판본을 통합한것으로 볼수 있다.

5판의 기본목적은 교육을 주는데 있다. 이 책에는 매장의 중요한 부분이나 측면들을 강조하기 위한 자료들이 추가되였다. 실례로 객체지향분석이나 객체지향설계와 같은 중요한 기법들을 개괄하고 있는 《진행과정》란들을 들수 있다.

이밖에 새로운 개요나 줄거리들이 학생들과 교원들에게 도움을 주게 된다. 또한 쏘프트웨어공학의 다양한 기법들을 수행하는 방법들에 대한 보충적인 자료들을 주기 위하여 이번 출판에서는 4판에서보다 실례연구를 더 자세히 서술하고 있다.

4판에는 《개발팀과 거래도구》라는 제목으로 된 장이 있다. 이번 출판에서는 교육학적견지로부터 자료들을 갱신하고 두 부분으로 나누어 그 기초를 이루고 있는 매개의 문제들을 보다 명백히 강조하고 있다. 특히 4장에서는 개발팀에 대하여 서술하고 있다면 5장에서는 쏘프트웨어공학자들이 리용하게 되는 도구들에 대하여 서술하고 있다.

이전과 마찬가지로 객체지향파라다임이 고전적인(구조화된)파라다임보다 우월함에도 불구하고 여기서는 고전적인 문제와 객체지향적인 자료들을 모두 포함시켜 주었다. 최신 쏘프트웨어공학교과서들에서는 객체지향파라다임만을 서술하고 고전적인 파라다임에 대하여서는 기껏해서 력사적인 고찰만을 서술해 주었다.

그러나 이 책에서는 경우가 다르다. 객체지향파라다임에 대한 요구가 높아 지고 고전적인 파라다임에 비한 그것의 우월성이 명백하게 증명되였음에도 불구하고 고전적인 파라다임에 대한 자료들을 포함시키는것은 본질적인 문제이다. 그 리유는 세가지 측면에서 설명할수 있다. 첫째로 고전적인 방법과 그리고 그것이 객체지향방법과 어떻게 차이나는가를 완전히 리해하지 않고서는 객체지향기술이 고전적인 기술보다 우월하다는것을 인식할수 없기때문이다. 둘째로 그 리유는 기술이전이 일반적으로 서서히 진행되기때문이다. 아직도 대다수의 쏘프트웨어개발기업체들에서는 객체지향파라다임을 받아 들이지 못하고 있다. 그러므로 이 책을 리용하게 되는 많은 대학생들은 고전적인 쏘프트웨어공학수법들을 리용하고 있는 개발기업체들에서도 일할수 있게 된다. 더우기 어떤 개발기업체가 새로운 쏘프트웨어를 개발하는데서 객체지향방법을 리용한다고 하더라도 현존 쏘프트웨어는 여전히 유지정비되여야 하는데 이러한 유산쏘프트웨어는 객체지향적이 아니다. 그러므로 고전적인 방법을 빼버리면 이 책을 읽게 되는 많은 대학

생들의 우려를 자아낼수 있다.

두 파라다임을 포함하게 된 리유는 셋째로 객체지향기술로 넘어 가려고 하는 개발기업체에서 일하게 되는 대학생은 개발기업체에 새 파라다임의 우점과 약점에 대하여 조언을 줄수 있게 되기때문이다. 그러므로 이전 판에서와 마찬가지로 여기서는 고전적인 방법과 객체지향적방법을 비교대조하고 분석하여야 한다. 4판은 통합모형화언어(UML)를 리용하는 최초의 쏘프트웨어공학교과서인데 UML에 대하여서는 그 책이 출판되기전에 이미 간단히 소개되였다. 지난 3년동안에 UML은 형식적으로 표준화되고 광범하게 리용되여 UML을 리용하지 않고 객체지향분석과 설계를 서술하고 있는 모든 교과서들은 인차 낡은것으로 되여 버렸다. 그러므로 도식을 리용하여 객체들과 그것들의 호상관계를 서술하는데서는 물론이고 객체지향분석과 객체지향설계를 진행하는데서 UML을 계속 리용하게 된다.

4판에서 소개된 또 하나의 화제는 설계패턴이였다. UML과 마찬가지로 설계패턴들은 현재 쏘프트웨어공학에서 주류를 이루고 있는 하나의 부분이다. 그렇기때문에 설계패턴과 관련한 자료는 의연히 중요하게 강조되고 있다.

이 책에서 또 한가지 새로운 화제는 최종적프로그람작성(*extreme programming*; XP)이다. XP에는 론의할 문제가 많지만 학생들이 XP에 대한 견해를 명백히 하도록 하는것이 중요하다. 그리하여 이 책에서는 그들스스로가 XP가 쏘프트웨어공학에서 돌파하여야 할 실제적인 전공대상이라는것을 인식하도록 하고 있다.

이전판에서는 문서작성과 유지정비관리, 재리용과 이식성, 시험과 콤퓨터지원쏘프트웨어공학(CASE)도구의 중요성이 강조되였다. 이 판에서도 이러한 모든 개념들이 다같이 강조되였다. 만약 그것들이 쏘프트웨어공학의 기초로 되는 중요한 문제라는것을 옳게 인식하지 못하면 학생들을 최신기술로 교육하는데 도움을 주지 못할것이다.

4판에서와 같이 객체지향생명주기모형, 객체지향분석, 객체지향설계, 파라다임에 대한 관리자측의 영향, 객체지향쏘프트웨어의 시험과 유지정비문제들에 깊은 주의를 돌려야 한다. 객체지향파라다임에도 역시 척도들이 포함되게 된다. 이밖에 객체에 대한 여러가지 보다 간단한 참고내용 즉 한 단락 또는 한 문장으로 된 참고내용들을 제시하고 있다. 그 리유는 바로 객체지향파라다임은 각이한 단계들이 수행되는 방법과 관련될뿐아니라 우리들이 쏘프트웨어공학에 대하여 생각하고 있는 방법들과 관련되여 있기때문이다. 이 책에서 객체기술이 폭 넓게 서술되고 있다.

쏘프트웨어개발공정은 여전히 이 책의 전반에 걸쳐 기초적인 개념으로 되고 있다. 공정을 조종하기 위하여서는 프로젝트에 대하여 무엇이 진행되고 있는가를 측정할수 있어야 한다. 따라서 여전히 척도문제가 강조되여야 한다. 공정개선에 주목하여 능력성숙도모형(CMM)과 CSO/IEEE 15504(SPECE)에 대한 자료들이 갱신되였고 ISO/IEEE 12207에 대한 자료가 추가되였다.

4판에서와 같이 여기서는 600여개의 참고문헌을 주었다. 여기서는 새롭고 련관된 고전적인 기사나 책들은 물론 현재까지 진행되여 온 연구론문들을 선택하여 주었다. 의심할바없이 쏘프트웨어공학은 급속히 발전하는 분야의 하나이기때문에 독자들이 최근의 결과들과 그것들을 어느 문헌을 찾아야 하는가를 알 필요가 있다. 이와 함께 오늘날의 최첨단연구가 어제날의 진리에 기초하고 있으며 어제날의 착상이 오늘날에 와서도 마찬가지로 적용될수 있다면 이전의 참고문헌들을 배제할 리유는 없다고 본다. 여기서는 독자

들이 Pascal, C, C++, Ada, Basic, COBOL, FORTRAN 또는 Java와 같은 고급언어들에 익숙하고 있으며 자료구조에 대한 과정안을 거쳤다는것을 전제로 하고 있다.

5판의 구성방법

5판은 4판에서와 같이 전통적인 한 학기분 과정안과 보다 새로운 두 학기분 과정안용으로 출판되였다. 전통적인 한 학기분 과정안에서는 교원이 학생들에게 리론적인 자료들을 속성교육하여 그들이 될수록 빨리 과정안상 목표에 도달하는데 필요한 지식과 기능을 가지도록 하였었다. 속성교육을 하는것은 학생들이 학기마감까지 과정안상 목표를 완성할수 있도록 그것에 인차 착수할수 있게 하기 위해서이다. 한 학기 즉 프로젝트에 기초한 쏘프트웨어공학과정안의 요구를 충족시키기 위하여 이 책의 2편에서는 단계별로 쏘프트웨어제품들의 생명주기를 고찰하며 1편에서는 2편을 리해하는데 필요한 리론적인 문제들을 서술하였다. 실례로 1편은 독자들에게 콤퓨터지원쏘프트웨어공학(CASE), 척도, 시험에 대하여 소개한다. 2편의 매장들에서는 매 단계를 위한 CASE도구, 척도, 시험에 대한 절들을 포함하고 있다. 교원이 해당 학기에서 2편을 인차 시작할수 있도록 1편을 간단히 취급하고 있다. 더우기 1편의 마지막 두 장(8장과 9장)은 뒤로 미루어 2편과 동시에 강의할수도 있다. 그러면 학생들은 될수록 빨리 과정안상 목표개척을 시작할수 있게 된다.

이제 두 학기분 쏘프트웨어공학과정안에 대하여 보기로 하자. 점점 더 많은 콤퓨터과학 및 콤퓨터공학학부들이 졸업생들의 절대다수가 콤퓨터기사로서의 직업을 찾고 있다는 사실을 깨닫고 있다. 따라서 많은 단과대학들과 종합대학들에서는 두 학기분 쏘프트웨어공학련속강의를 계속하고 있다. 첫번째 과정안은 대체로 리론적인 내용을 취급하고 있다(그러나 거의 언제나 어떤 종류의 하나의 작은 프로젝트로 된다.). 두번째 과정안은 주요개발팀에 의한 과정안상 목표 즉 일반적으로 가장 고급한 프로젝트로 이루어 진다. 과정안상 목표가 두번째 과정안으로 될 때 교원은 2편의 강의시작을 서두를 필요가 없게 된다. 그러므로 5판을 리용하여 한 학기련속강의를 하는 교원은 제1장부터 제7장까지의 대부분의 내용을 강의하고 그다음에 2편(제10장부터 제16장까지)강의를 진행하게 된다. 그다음 제8장과 제9장은 학생들이 과정안상 목표개척을 진행하는동안 제2편과 병렬로 또는 과정안의 마감에 강의할수 있다. 두 학기 련속강의를 할 때에는 이 책의 장순서대로 진행하면 된다. 그러면 학생들은 다음학기에서 진행하게 되는 개발팀에 의한 과정안상 목표개척을 완성할수 있다.

2편의 주요 쏘프트웨어공학기법들을 옳게 리해시키기 위하여 매 기법들이 두번씩 서술되고 있다. 처음에는 기법들을 소개할 때마다 승강기문제를 통하여 설명해 준다. 승강기문제는 독자들이 완전한 문제에 적용되는 기법들을 고찰할수 있도록 하는데 적당한 문제로 되며 해당한 기법의 우점과 약점을 모두 강조하는데 충분하다. 그다음 새로운 실례연구와 관련한 부분들을 매장의 마감에 주었다. 이러한 세부적인 해결방안들은 매 기법에 대한 두번째 례증을 주는것으로 된다.

문제설정

이전판에서와 마찬가지로 네가지 류형의 문제들이 존재한다.

첫째로 매장의 마감부분에 중점들을 강조한 문제들을 주었다. 이 문제내에는 매개 문제를 풀기 위한 기술정보들이 포함되여 있다.

둘째로 쏘프트웨어과정안상 목표가 있다. 그것은 일반전화를 사용하여 서로 의논을 진행할수 없는 세명의 학생으로 이루어 진 개발팀이 해결할수 있도록 설계되였다. 과정안상 목표는 16개의 개별적인 구성요소들로 이루어 져 있는데 매개 구성요소들은 련관된 장들과 결합되여 있다. 실례로 13장의 주제는 설계이다. 그래서 이 장에서는 과정안상 목표를 쏘프트웨어설계와 관련된것들로 구성하여 주었다. 큰 프로젝트는 보다 작은 부분들로 쪼개여 학생들이 더 잘 리해하고 해결해 나갈수 있도록 하였다. 과정안상 목표의 구조는 교원이 16개의 구성요소들을 자기가 선택한 임의의 다른 프로젝트들에 자유롭게 적용할수 있도록 되여 있다.

셋째로 이 책은 상급반학생들은 물론 졸업한 학생들이 리용할수 있도록 씌여 졌기때문에 쏘프트웨어공학문헌에 있는 연구론문들에 기초하고 있다. 매장에서는 중요한 론문들을 선택하여 주었고 객체지향쏘프트웨어공학과 관련한 론문들도 주었다. 학생들은 론문을 읽으면서 물어도 보고 그 내용과 관련된 질문들에 대답도 하게 된다. 물론 교원은 기타 다른 연구론문들을 마음대로 배당해 줄수 있다. 즉 매장의 마감부분에 있는 《보충》에서는 련관된 론문들에 대하여 폭 넓고 다양하게 소개하고 있다.

네번째 류형의 문제는 실례연구와 관련한것들이다. 이 류형의 문제들은 제품을 처음부터 개발하는것보다 현존제품을 수정하는 방법을 교육하면 학생들이 더잘 배울수 있다는 관점으로부터 3판에서 처음으로 도입되였다. 쏘프트웨어산업에 종사하는 많은 중간급쏘프트웨어공학자들이 이 견해에 공감을 표시하고 있다. 따라서 실례연구를 서술한 매장들에는 학생들이 같은 방법으로 실례연구를 수정하는데 필요되는 적어도 세가지 문제들을 포함하고 있다. 실례로 어떤 장에서는 학생들이 그 실례연구에서 리용한것과는 다른 설계기법을 리용하여 실례연구를 재설계할것을 제기하고 있다. 또 다른 장에서는 학생들이 다른 순서로 객체지향분석단계를 수행하게 되면 어떤 결과가 발생하게 되는가 하는것을 제기하고 있다. 실례연구에 있는 원천코드를 보다 쉽게 수정하기 위하여서는 World Wide Web상에서 홈페지 **www.mhhe.com/engcs/ compsci/schach**를 리용할수 있다.

web싸이트는 또한 완전한 PowerPoint강의록은 물론 이 책에 있는 모든 그림들에 대하여 명백히 정통할수 있게 한다.

교원들의 해답지도서는 과정안상 목표에 대하여서는 물론 모든 련습에 대한 자세한 해답을 주었다. 교원들의 해답지도서는 McGraw-Hill에서 얻을수 있다.

제 1 편

쏘프트웨어공학에 대한 개괄

이 책에서 제1편에 포함된 9개의 장들은 이중적인 역할을 놀고 있다. 이 장들에서는 독자들에게 쏘프트웨어개발공정에 대하여 소개하며 이 책의 개괄에 대하여 서술하고 있다. 쏘프트웨어개발공정은 쏘프트웨어를 개발하는 과정이다. 제품은 착상으로 시작하여 마지막에 페기로 끝난다. 이 과정에 제품은 요구사항확정과 명세작성, 설계, 실현, 통합, 유지정비, 최종적으로 페기 등과 같은 단계를 통하여 개발된다. 쏘프트웨어개발공정에는 또한 쏘프트웨어를 개발하고 유지정비하는데 리용하는 도구들과 기법들이 포함된다.

제1장 《쏘프트웨어공학의 범위》는 쏘프트웨어제품을 위한 기법이 비용상 효과적이여야 하며 쏘프트웨어제품개발팀성원들사이에서 건설적인 호상작용을 촉진시켜야 한다는데 대하여 지적하고 있다. 이 책 전반에서는 객체의 중요성을 강조하고 있다.

제2장의 제목은 《쏘프트웨어개발공정》이다. 여기서 개발공정의 매 단계들에 대하여 상세히 론의한다. 이 장에서는 쏘프트웨어공학의 많은 문제들이 론의되지만 그 해결방안에 대하여서는 제시하지 않는다. 대신에 독자들에게 매 문제들이 이 책의 어느 부분에서 고찰되는가를 알려 준다. 이 장은 이 책의 나머지부분들에 대한 안내서로 된다. 이 장은 쏘프트웨어개발공정의 개선에 관한 문제들의 고찰로 결속된다.

제3장 《쏘프트웨어생명주기모형》에서는 여러가지 서로 다른 쏘프트웨어생명주기모형들을 상세히 론의하고 있다. 여기서는 폭포모형, 신속원형작성모형, 증식모형, 최종적프로그람작성, 동기 및 안정화모형, 라선모형들을 서술한다. 독자들이 특정한 프로젝트에 대하여 적당한 생명주기모형을 결정할수 있도록 여러가지 생명주기모형들을 비교하였다.

제4장의 제목은 《개발팀》이다. 오늘날 프로젝트는 규모가 아주 커서 주어 진 시간내에 개별적으로 완성할수 없다. 그리하여 쏘프트웨어를 전문으로 하는 개발팀들에서는 프로젝트연구에서 서로 협력하여 실현하고 있다. 이 장에서 론의하게 되는 중요한 론점은 개발팀성원들이 제품개발에 함께 참가하도록 개발팀을 어떻게 조직하겠는가 하는것이다. 개발팀을 조직하는 방법들로서는 민주적인 개발팀과 책임프로그람작성자개발팀, 동기 및 안정화개발팀 그리고 최종적인 프로그람작성팀들을 포함한 여러가지 방법들이 있다.

제5장에서는 《거래도구》를 론의하고 있다. 쏘프트웨어공학자들은 리론적으로나 실천적으로 여러가지 많은 도구들을 리용할수 있어야 한다. 이 장에서는 여러가지 다양한 쏘프트웨어공학도구들을 고찰하게 된다. 이러한 도구들중의 하나는 큰 문제를 보다 작은 문제로, 보다 다루기 쉬운 문제로 분해하여 실현하는 기술인 계단식세련이다. 또 다른 도구는 비용 대 리득분석인데 이 기술은 쏘프트웨어프로젝트가 재정적으로 실현가능한가를 결정하는 기술이다. 그다음 콤퓨터지원쏘프트웨어공학(CASE)도구가 고찰된다. CASE도구

는 쏘프트웨어공학자들로 하여금 쏘프트웨어를 개발하고 유지정비하는데 도움을 줄수 있는 쏘프트웨어제품이다.

마지막으로 쏘프트웨어개발공정을 관리하기 위하여 프로젝트가 제 궤도에 있는가 하는것을 결정하는 여러가지 량들을 측정하는것이 필요하다. 이러한 측도(척도)는 프로젝트의 성공에서 결정적인 작용을 한다. 제5장의 마지막 두개의 론점인 CASE도구와 척도에 대하여 제10장부터 제16장까지에서 상세히 취급하는데 여기서는 쏘프트웨어생명주기의 특정한 단계들에 대하여 서술한다. 매 단계를 적절하게 관리하는데 필요한 척도는 물론 그 단계를 지지하는 CASE도구에 대하여서도 론의한다.

이 책에서 한가지 중요한 론점은 시험이 제품이 의뢰자에게 배포되기전에 지어 쏘프트웨어생명주기의 매 단계의 마감에서 수행되게 되는 분리된 단계가 아니라는것이다. 그 대신 시험은 모든 쏘프트웨어생산활동과 동시에 진행되게 된다.

제6장 《시험》에서는 시험의 기초를 이루고 있는 개념들이 론의된다. 쏘프트웨어생명주기의 개별적인 단계에 대한 특정한 시험기법들을 제10장부터 제16장까지에서 언급한다.

제7장의 제목은 《모듈로부터 객체에로》이다. 여기서는 클라스와 객체 그리고 객체지향파라다임이 구조화파라다임에 비하여 보다 성공적이라고 하는것과 관련한 상세한 설명이 제시된다. 이 장의 개념들은 이 책의 나머지부분 특히 제12장 《객체지향분석단계》와 객체지향설계를 서술하고 있는 제13장 《설계단계》등에서도 리용된다.

제7장의 기본착상은 제8장 《재리용성, 이식성, 호상조작성》에로 확장된다. 중요한 것은 서로 다른 여러가지 하드웨어상에서 정확히 동작할수 있으며 의뢰기-봉사기와 같은 분산적인 구성방식에서도 동작할수 있는 재리용가능한 쏘프트웨어를 작성할수 있는것이다. 이 장의 첫 부분에서는 재리용에 대하여 서술한다. 즉 객체지향패턴과 기본구성과 같은 재리용전략은 물론 재리용의 다양한 실례연구들이 포함된다. 이식성은 두번째 문제점이다. 즉 이식성에 대하여 약간 깊이 있게 서술한다. 이 장은 CORBA와 COM과 같은 호상조작성문제들로서 결속된다.

이 장에서 다시 생각해 볼 문제는 재리용성, 이식성 그리고 호상조작성을 실현하는데서 객체의 역할에 관한 문제이다.

제1편의 마감장은 제9장 《계획작성과 타산》이다. 어떤 쏘프트웨어프로젝트를 시작하기에 앞서 전체 조작을 구체적으로 계획하는것이 본질적인 문제이다. 일단 프로젝트를 시작하면 관리자는 계획으로부터의 탈선을 포착하고 필요한 정정작업을 수행하면서 개발공정을 조종하여야 한다. 또한 의뢰자에게 프로젝트를 완성하는데 얼마만한 시간과 품이 드는가 하는데 대한 정확한 평가를 주어야 한다. 여기서 기능점수와 COCOMO Ⅱ를 비롯한 여러가지 타산기법들이 주어 진다. 또한 쏘프트웨어프로젝트관리계획에 대한 상세한 설명도 준다. 이 장에서 언급한 내용들은 제11장과 제12장에서 리용된다. 고전적인 파라다임을 리용할 때 기본계획작성과 타산활동은 제11장에서 설명한바와 같이 명세작성단계의 마감부분에서 진행된다. 객체지향파라다임을 리용하여 쏘프트웨어를 개발할 때에는 이러한 계획작성이 객체지향분석단계의 마감에서 진행된다(제12장).

제１장. 쏘프트웨어공학의 범위

콤퓨터가 계산해 낸 0.00딸라의 계산서를 받은 경영자에 대한 이야기는 널리 알려져 있다. 경영자는 《멍텅구리콤퓨터》에 대하여 친구들로부터 조소를 받고 나서 계산서를 버리였다. 한달후에 류사한 계산서가 도착하였는데 이때 계산서에는 30일이 표시되여 있었다. 그다음 세번째 계산서가 왔다. 네번째 계산서가 한달후에 또 왔는데 그 계산서와 함께 0.00딸라에 대한 계산서를 당장 물지 않으면 가능한 법적행동으로 넘어 가겠다는것을 암시한 전보문이 왔다.

120일로 표식된 5번째 계산서에는 그 어떤 암시도 없었다. 즉 이러한 미치광이기계에 의하여 자기 기업체의 신용이 떨어 질가봐 두려워 경영자는 쏘프트웨어공학자인 친구를 불러 불쾌한 이야기를 모두 말해 주었다. 웃지 않으려고 하면서 그 쏘프트웨어공학자는 0.00딸라에 대한 계산서를 우편으로 보내라고 경영자에게 이야기해 주었다. 이것은 바라던 효과를 보았으며 그로부터 며칠후에 0.00딸라에 대한 수납이 되였다. 경영자는 앞으로 콤퓨터가 여전히 0.00딸라를 빚지고 있다고 주장할수 있는 경우를 고려하여 그것을 조심스럽게 서류철에 넣어 두었다. 널리 알려 진 이 이야기의 몇가지 속편들은 잘 알려져 있지 않다. 며칠후에 경영자는 자기의 은행관리인에게 호출되였다. 은행관리인은 행표를 보여 주면서 《당신의 행표입니까?》하고 물었다. 경영자는 그렇다고 하였다. 《당신은 왜 0.00딸라에 대한 계산서를 썼는지 나에게 말해 주겠습니까? 》라고 물었다.

그래서 그 이야기전부가 되풀이되였다.

경영자가 이야기를 끝내자 은행관리인은 그에게 조용히 물었다.

《당신은 0.00딸라에 해당한 계산서가 우리 콤퓨터체계에서 무엇을 하였는지 생각해 보았습니까?》

어떤 콤퓨터전문가들은 이 이야기를 듣고 웃을수 있다.

결국 우리모두는 그 원래의 형태에서 0.00딸라에 대한 글자들에도 지불을 독촉하는 결과를 빚어 낸 그러한 제품을 설계 또는 실현하고 있다.

지금까지 개발자들은 언제나 시험단계에서 이러한 오유를 범하군 하였다. 그러나 이것을 비웃는것도 의미가 있다. 왜냐하면 개발자들은 그 제품이 손님들에게 배포되기전에 얼마간은 오유들을 발견하지 못할수 있다고 하는 우려를 하고 있기때문이다.

그리 유모아적이라고 할수 없는 한가지 쏘프트웨어오유는 1979년 11월 9일에 발견되였다. 전략공군사령부는 전 세계적군사명령 및 조종체계(WWMCCS)콤퓨터망이 이전 쏘련이 미국을 겨냥한 미싸일을 발사하였다는것을 보고하자 매우 놀라게 되였다[Neumann, 1980]. 사실은 바로 약 5년후에 나온 전쟁유희영화에서와 같이 모의공격을 실제적인것으로 해석하였던것이다. 비록 국방성이 실제자료에 시험자료를 넣음으로써 정확한 기구에 대한 세부를 주지 않았다고 할지라도 이 문제를 쏘프트웨어오유에 기인시키는것은 타당할것 같다. 전체 체계가 모의와 현실을 구별할수 있도록 설계되지 않았거나 또는 사용자

대면부에 체계의 말단사용자들이 사실을 가정과 구별할수 있다는것을 담보하기 위한 필요한 검사를 포함시키지 않았다. 달리 말하여 만일 문제가 쏘프트웨어에 의하여 발생하였다면 쏘프트웨어오유는 우리들로 하여금 불쾌하고도 불의적인 결말을 리해하는것과 같은 결과를 가져 올것이다(쏘프트웨어오유에 의하여 발생되는 다른 재난에 대한 정보를 알려면 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

WWMCCS망의 경우에 재난은 제일 마지막 1min내에 방지된다. 그러나 다른 쏘프트웨어오유에 대한 결과는 때때로 비극적이다. 실례로 1985년과 1987년사이에 적어도 두명의 환자가 Therac-25의학적선형가속기에서 나오는 방사선의 한계량초과의 결과로 죽게 되였다. 그 원인은 조종쏘프트웨어의 오유에 있었다.

1991년 만전쟁시기 스쿠드미싸일이 패트리오트요격미싸일방어를 뚫고 사우디아라비아의 다흐란가까이에 떨어 졌다. 미군병사 28명이 죽고 98명이 부상을 당했다. 패트리오트미싸일의 쏘프트웨어는 루적시간오유를 포함하고 있었다. 패트리오트미싸일은 한번에 몇시간동안만 동작하고 그다음에 시계가 재설정되도록 설계되였다.

결국 오유는 크게 나타나지 않고 발견되지도 않았다.

만전쟁시기 다흐란에서 패트리오트미싸일전원은 100h이상 계속 동작하였다.

이것은 루적된 시간불일치가 체계가 정확하게 동작할수 없을 정도로 크게 하였다. 만전쟁시기 미국은 스쿠드미싸일을 방어하기 위하여 패트리오트미싸일을 배에 실어 이스라엘로 수송하였다. 8h만에 시간계수문제를 발견한 이스라엘은 즉시에 그에 대하여 미국의 제작자들에게 통보하였다. 제작자들이 오유를 가능한껏 빨리 퇴치하였지만 비극적으로 스쿠드미싸일의 타격을 받은 다음에야 새로운 쏘프트웨어가 도착하였다[Mellor, 1994].

우리가 계산서문제나 반항공방어를 취급한것은 많은 쏘프트웨어들이 잔류오유를 가지며 예산을 초월하여 늦게 배포되게 되기때문이다. 쏘프트웨어공학은 이 문제를 해결하기 위하여 노력하였다. 달리 말하면 쏘프트웨어공학은 사용자들의 요구를 만족시키는 오유 없는 쏘프트웨어를 주어 진 기간과 예산으로 생산하는것을 목적으로 하고 있는 학문이다. 더우기 쏘프트웨어는 사용자들의 요구가 달라 질 때 보다 쉽게 수정할수 있어야 한다. 이 목적을 달성하기 위하여 쏘프트웨어공학자들은 기술적 및 관리적측면에서 폭 넓은 기능을 소유하여야 한다. 이러한 기능은 프로그람작성뿐아니라 요구사항확정으로부터 유지정비에 이르는 쏘프트웨어생산의 매 단계에 적용되여야 한다.

쏘프트웨어공학의 범위는 아주 넓다. 쏘프트웨어공학의 일부 측면들은 수학이나 콤퓨터과학에서와 같이 분류될수 있으며 다른 측면들은 경제적, 관리적 및 심리학적인 측면에 귀착된다. 쏘프트웨어공학의 범위를 넓히기 위하여 5개의 각이한 측면들을 시험해 보기로 한다.

1. 1. 력사적인 측면

사실 발전기가 고장나기는 하지만 그것은 로임지불명단관리제품에서보다는 훨씬 적다. 또한 때때로 다리들이 무너지지만 조작체계의 붕괴보다는 훨씬 적다. 쏘프트웨어의 설계와 실현, 유지정비가 전통적인 공학학문들과 같은 지반을 가지고 있다는 확신에 기초하여 1967년에 NATO연구그룹이 쏘프트웨어공학이라는 용어를 만들어 냈다.

쏘프트웨어의 구축이 다른 공학적인 과제들과 류사하다고 하는 주장은 도이췰란드의 가르미슈(Garmisch)에서 열린 1968 NATO쏘프트웨어공학대회에서 찬동을 받았다. 이러한 찬동은 그리 놀라운것은 아니다. 즉 대회라는 명칭이 쏘프트웨어생산이 공학적인 활동으로 되여야 한다는 믿음을 반영해 주고 있다.

대회참가자들이 내린 결론은 공학적인 학문의 원리와 파라다임을 리용하여 그들이 명명한 쏘프트웨어의 위기문제를 해결하여야 한다는것이다. 이러한 위기는 바로 쏘프트웨어품질이 일반적으로 접수할수 없을 정도로 낮으며 최종기한과 비용의 한계가 맞아 떨어 지지 않는다는것이였다. 많은 쏘프트웨어의 성공에 대한 이야기들이 있음에도 불구하고 상당한 량의 쏘프트웨어는 여전히 뒤늦게 예산을 초월하여 그리고 오유를 포함한 채로 배포되고 있다.

쏘프트웨어위기가 30년이 지난 오늘도 여전히 남아 있다는것은 두가지 문제점을 말해 주고 있다. 첫째로 쏘프트웨어생산공정은 많은 측면에서 전통적인 공학과 류사하지만 그자체의 고유한 특성과 문제점을 가지고 있다. 둘째로 쏘프트웨어위기는 그것이 장기성을 띠며 예측하기 어렵다는 관점으로부터 아마도 쏘프트웨어불경기라고 바꾸어 불러야 할것 같다. 확실히 다리는 조작체계만큼 자주 무너 지지는 않는다. 그러면 왜 다리건설기술이 조작체계구축에 리용될수 없는가? NATO대회에서 스쳐 지나버린것은 분필이 치즈와 다른것처럼 다리가 조작체계와는 다르다는 사실이다.

다리와 조작체계사이의 기본차이는 무너져 내리는 동작에 대한 일반공학자들의 집단과 쏘프트웨어공학자들의 집단의 관점에 있다. 1940년에 무너져 내린 타코마의 좁은 다리와 마찬가지로 다리가 무너져 내리면 다리는 거의 모두 다시 설계하고 파괴된것을 다시 세워야 하였다. 원래의 설계는 잘못 된것이였고 인간의 안정을 위협하는것으로 하여 설계는 철저히 바꾸어 지게 된다. 이밖에 거의 모든 실례들에서 다리가 무너져 내린 결과 다리구조물에 너무 많은 손상을 주기때문에 유일하게 합리적인 방도는 파괴된 다리의 잔해를 파괴해 버리고 그것을 완전히 다시 설계하고 건설하는것이다. 더우기 같은 설계로 건설된 다른 다리들은 주의 깊게 검사되여야 하며 최악의 경우에는 설계를 다시 하고 다시 건설하여야 한다. 조작체계의 폭주는 범상치 않은 일로서 간주되며 드물게는 그 설계에 대한 즉시적인 조사를 진행하게 한다. 폭주가 발생하면 그 폭주를 초래한 조작환경이 재발하지 않도록 하기 위하여 간단히 체계를 다시 초기화하는것이 가능할수도 있다. 흔히 있는 경우이지만 폭주원인에 대한 근거가 밝혀 지지 않은 경우에는 그것이 유일한 보수대책으로 될수 있다. 폭주로 인한 손해는 보통 어떤 자료기지가 부분적으로 손상되였든가 일부 파일이 잃어 진것과 같은 부차적인것들이다. 지어 파일체계에 대한 손상이 크다면 흔히 여분의 자료는 파일체계를 선행한 폭주조건을 완전히 제거하지 못한 어떤 상태로 체계를 회귀할수 있다. 아마도 일반공학자들이 다리의 파괴를 취급하는것처럼 쏘

프트웨어공학자들이 조작체계를 심중하게 취급하면 쏘프트웨어공학에서의 전문화수준이 자연적으로 올라 가게 될것이다.

이제 실시간체계 즉 실세계로부터 들어 오는 입력이 발생하자마자 곧 그 입력에 응답할수 있는 체계를 고찰해 보자. 한가지 실례로서 콤퓨터조종집중치료기를 들수 있다. 사실상 아무리 많은 구급환자들이 동시에 발생한다고 하여도 체계는 상태가 위급하지만 안정한 환자들을 감시하는것을 중단함이 없이 새로운 구급환자들에 대하여 주의를 돌릴것을 의료일군들에게 계속 경고해야 한다. 일반적으로 집중치료기나 핵반응로, 우주정류소에서의 기후조건들을 조종하는것과 같은 실시간체계에서의 고장은 그 영향이 아주 크다. 그러므로 대부분의 실시간체계들은 고장의 영향을 최소화하기 위하여 일련의 오유허용한계를 포함하고 있다. 즉 체계는 임의의 고장을 자동적으로 회복할수 있도록 설계된다.

바로 오유허용한계에 대한 개념은 다리와 조작체계사이의 기본차이를 강조하고 있다. 다리들은 강한 바람, 불의의 큰물 등과 같은 상당히 예측하기 어려운 조건들에 견디여낼수 있도록 설계된다.

대부분의 쏘프트웨어작성자들이 전제로 하고 있는 하나의 절대적인 가정은 쏘프트웨어가 극복해야 할 가능한 모든 조건들을 예측할수 없으므로 예견하지 못한 조건으로부터 초래되는 피해를 최소화하도록 쏘프트웨어를 설계하여야 한다는것이다.

달리 말하면 다리는 완전하게 설계된다고 가정하며 반대로 대부분의 조작체계는 불완전하게 설계된다고 가정한다. 즉 대부분의 조작체계들은 재초기화조작이 사용자들이 필요할 때마다 할수 있는 간단한 조작으로 되는것처럼 그러한 방식으로서 설계된다. 이러한 차이가 오늘날 그처럼 많은 쏘프트웨어를 왜 공학적으로 구성되였다고 볼수 없는가 하는 기본리유로 된다.

이와 같은 차이는 다만 일시적이라고 볼수 있다. 결국 우리는 다리를 수천년동안 세워 왔으며 다리가 견디여 내야 할 각이한 류형의 조건들에 대하여 충분한 경험과 전문기술을 가지게 되였다. 우리는 조작체계에 대하여 다만 50여년간의 경험을 가지고 있을뿐이다. 우리가 더 많은 경험을 가지게 되면 명백히 우리가 다리를 리해하는것과 마찬가지로 조작체계를 리해하게 될것이며 결국은 오유가 없는 조작체계를 구성할수 있게 될것이라는 론리가 성립한다. 이러한 론의에서 약점은 하드웨어의 복잡성 그리고 그로 인하여 조작체계의 복잡성이 그것에 대한 파악가능성보다 더 빨리 증대된다는것이다. 1960년대에는 다중프로그람작성조작체계를 가지게 되였고 1970년대에는 가상기억을 취급하였으며 이제는 다중처리기가 분산(망)조작체계에 자리를 양보하려 하고 있다. 우리가 조작체계와 같은 쏘프트웨어제품을 구성하는 여러가지 구성요소의 호상결합으로 하여 생기는 복잡성을 조종할수 있기전에는 그것을 완전히 리해할수 없다. 즉 리해하지 못하면 그것을 설계할수 없다.

쏘프트웨어의 복잡성의 원인은 그것이 실행될 때 불련속적인 상태로 동작한다는것이다. 지어 한비트만 변하여도 쏘프트웨어는 상태변화를 일으킨다. 이러한 상태들의 총수는 굉장히 많으므로 그 대부분의 상태를 개발팀에서 다 고려하지는 못하였다. 만일 쏘프트웨어가 이런 예측하지 못한 상태에 놓이게 되면 그 결과는 흔히 쏘프트웨어의 고장으로 끝난다. 다리는 련속적인(상사적인) 체계이다. 그것들은 련속수학, 본질적으로는 계산을 리용하여 서술된다. 그러나 조작체계와 같은 불련속체계는 리산수학을 리용하여 서술되

여야 한다[Parnas, 1990]. 그러므로 쏘프트웨어공학자들은 이와 같은 복잡성을 처리하려고 할 때 하나의 기본도구로 되는 리산수학에 정통하여야 한다.

다리와 조작체계의 두번째 기본차이는 유지정비이다. 어떤 다리를 유지정비하는것은 일반적으로 그것을 장식하며 갈라 진 틈을 수리하고 도로를 다시 닦는것 등에 제한되여 있다. 일반공학자들에게 다리를 90° 회전시켜 달라고 하거나 185km 이동시켜 달라고 하면 그들은 무지막지한 요구라고 생각할수 있다. 그러나 묶음식조작체계로부터 시간분할조작체계에로 변환하거나 한 콤퓨터로부터 완전히 다른 구성특성을 가진 다른 콤퓨터로 체계를 이식하기 위하여 쏘프트웨어공학자들을 요청할 일은 전혀 없는것 같다. 만일 어떤 조작체계를 새로운 하드웨어에 이식하는 경우에 그 조작체계 원천코드의 50%를 5년주기로 재작성하는것은 보통일이다. 그러나 공학자들중에서 다리의 절반을 교체할수 있다고 생각하는 사람은 없을것이다. 즉 안전요구사항들에는 새 다리를 세울것을 지적하고 있다. 그러므로 유지정비령역은 쏘프트웨어공학이 전통적인 공학과 차이나는 두번째의 기본측면으로 된다. 쏘프트웨어공학에 대한 그밖의 유지정비측면에 대하여서는 1.3에서 서술하고 먼저 경제적측면들에 대하여 고찰하기로 한다.

1. 2. 경제적인 측면

화학공학과 화학사이의 관계를 비교함으로써 쏘프트웨어공학과 콤퓨터과학사이의 관계에 대한 견해를 가질수 있다. 결국 콤퓨터과학과 화학이 모두 과학이라는것은 그것들이 리론적인 구성요소와 실천적인 구성요소를 가지고 있기때문이다. 화학의 경우에 실천적인 구성요소는 연구사업으로 되지만 콤퓨터과학의 경우에 실천적인 구성요소는 프로그람작성으로 된다.

석탄으로부터 휘발유를 추출하는 공정을 생각해 보자. 제2차세계대전기간에 도이췰란드는 연유공급이 단절되였기때문에 자기들의 전쟁무기용연료를 만들기 위하여 이 공정을 리용하였다.

인종차별을 반대하는 원유봉쇄가 실시되는 동안 남아프리카공화국정부는 《남아프리카의 석탄을 원유로》(South African Coal into Oil, SASOL)라는 계획밑에 1조딸라를 쏟아 부었다. 남아프리카액체연료수요의 약 절반이 이 방법으로 충당되였다.

어떤 화학자의 견지에서 보면 석탄에서 휘발유를 얻는데는 여러가지 방법들이 있는데 다같이 중요하다. 결국 그 어떤 화학반응도 다른 화학반응보다 더 중요하지는 않다. 화학공학기사들의 견지에서 보면 어느 한 때 석탄으로부터 휘발유를 합성해 내는 하나의 중요한 수단 즉 경제적으로 가장 인기를 끄는 하나의 반응은 명백히 존재한다. 달리 말하면 화학공학기사들은 가능한 모든 반응들에 대하여 평가하고 그중에서 1*l*당 비용이 가장 낮은 반응을 제외하고는 모두 버리는것과 같다.

콤퓨터과학과 쏘프트웨어공학사이에도 이와 같은 류사한 관계가 있다. 콤퓨터과학자들은 좋은것과 나쁜것을 모두 포함하여 쏘프트웨어를 생산하기 위한 여러가지 방법들을 연구한다. 그러나 쏘프트웨어공학자들은 경계적가치가 있는 그러한 기법들에만 흥미를 가진다.

실례로 현재 코드작성기법 CT_{old}을 리용하고 있는 한 쏘프트웨어기업체는 새로운 코드작성기법 CT_{new}가 CT_{old}에 필요한 시간의 9/10로 따라서 9/10의 비용으로 코드를 생성한다는것을 발견하였다. CT_{new}기술이 리용하기 적합한 기법이라는것이 공통적인 인식으로 되는것 같다. 사실상 선택기법이 보다 빠른 기법이라는것이 공통적인 인식인것 같지만 쏘프트웨어공학의 경제학은 그 반대사실을 나타낼수도 있다. 그 하나의 리유는 어떤 기업체가 새로운 기술을 도입할 때 드는 비용으로 설명할수 있다.

CT_{new}기법을 리용하게 되면 코드작성이 10% 더 빨라 진다고 하는 사실은 CT_{new}기법을 도입하는데 드는 비용에 비하면 그리 중요하지 않을수도 있다. 양성에 필요한 비용을 보상하기전에 2～3개의 프로젝트를 완성하는것이 필요할수도 있다. 또한 CT_{new}과정안을 집행하는 동안은 프로그람작성자들이 생산활동에 종사할수 없게 된다. 비록 배우고 돌아왔다 해도 짧은 기간에 급하게 배워 왔으므로 쏘프트웨어전문가가 CT_{old}와 마찬가지로 CT_{new}에 숙련되자면 몇달동안은 CT_{new}를 련습하여야 할수도 있다. 그러므로 CT_{new}를 리용하여 첫 프로젝트를 완성하는데는 CT_{old}를 리용할 때보다 더 오랜 시간이 걸릴것이다. CT_{new}로 전환하겠는가를 결정할 때 이러한 비용에 관한 문제를 고려해야 한다.

무엇때문에 쏘프트웨어공학의 경제적측면이 CT_{old}가 유지되여야 한다는것을 암시할수 있는가 하는 두번째 리유는 바로 유지정비의 결과이다. CT_{new}코드작성기술은 실지 CT_{old}에 비하여 10% 더 빠를수 있다. 그 결과에 생성된 코드는 의뢰자들의 현재의 수요를 만족시킬수 있는 품질을 가질수 있다. 그러나 CT_{new}기술을 리용하면 제품의 사용기간보다도 CT_{new}값을 더 높게 설정함으로써 유지정비하기 어려운 코드를 생성할수 있다. 물론 쏘프트웨어개발자들은 유지정비에 아무런 책임도 없다는 관점으로부터 CT_{new}가 가장 매력이 있는 제안으로 된다. 결국 CT_{new}의 리용은 비용을 10%로 더 적어 지게 한다. 의뢰자들은 CT_{old}기술을 리용할것을 주장하며 따라서 쏘프트웨어의 전체적인 사용기간에 드는 비용이 더 적어 질것이라는 기대를 가지고 보다 더 높은 초기비용을 지불하여야 한다. 의뢰자와 쏘프트웨어제작자들의 유일한 목적은 될수록 빨리 코드를 작성하는것이다. 일반적으로 어떤 특정한 기법을 리용하는데 오랜 시간이 드는것은 단기리득의 리윤에 의하여 무시된다. 쏘프트웨어공학에 경제적인 원리들을 적용하는것은 의뢰자들이 장기비용을 감소시키는 기술을 선택할것을 요구한다.

1. 3. 유지정비적인 측면

개념의 구상으로부터 마지막 페기에 이르기까지 쏘프트웨어가 거치게 되는 매 단계들을 생명주기(*life cycle*)라고 부른다. 이 기간에 제품은 여러단계 즉 요구사항확정, 명세작성, 설계, 실현, 통합, 유지정비, 페기단계를 거치게 된다. 생명주기모형에 대하여 제3장에서 보다 구체적으로 론의한다. 여기에서는 유지정비의 개념을 정의하기 위한 문제들을 론의한다.

1970년대 말까지 대부분의 개발기업체들은 현재 폭포모형(*waterfall model*)이라고 부르고 있는 생명주기모형을 리용하여 쏘프트웨어를 개발하여 왔다. 이 모형에 대한 여러가지 변종들이 있는데 제품은 크게 7개의 기본적인 단계를 거쳐 개발된다. 이러한 단계들

은 어떤 특정한 기업체의 단계들에 정확하게 대응하지 않을수도 있지만 이 책이 목적하고 있는 실천에는 충분히 가깝다. 이와 류사하게 매 단계들에 대한 정확한 이름도 기업체에 따라 달리 불리우고 있다. 여러단계들에 리용되는 이름들은 독자들이 잘 리해할수 있도록 될수록 일반적인것으로 선택하였다. 그림 1-1에 매 단계들을 개괄하여 주고 그 단계들이 제시되는 장들도 표시하여 주었다.

1. 요구사항확정단계 (10장)
2. 명세작성(분석)단계 (11장, 12장)
3. 설계단계 (13장)
4. 실현단계 (14장, 15장)
5. 통합단계 (15장)
6. 유지정비단계 (16장)
7. 페기

그림 1-1. 쏘프트웨어생명주기의 단계와 해당한 장들

1. 요구사항확정단계(*Requirement phase*) 이 단계에서는 개념들을 찾아 내고 엄밀히 규정하며 의뢰자들의 요구를 도출해 낸다.

2. 명세작성(분석)단계(*Specification*(*analysis*) *phase*) 이 단계에서는 의뢰자들의 요구를 분석하고 그것을 《이 제품은 무엇을 하기로 제안되는가?》라는 명세서형식으로 제시한다. 이 단계를 때때로 분석단계라고도 한다. 이 단계의 마감에서 제안된 쏘프트웨어개발을 아주 상세히 서술하고 있는 쏘프트웨어프로젝트관리계획(*software project management plan*)을 작성한다.

3. 설계단계(*Design phase*) 명세서는 두개의 련속적인 설계과정을 거치게 된다. 첫째로 구성방식설계가 있는데 여기서는 제품을 모듈(*module*)이라고 부르는 구성요소들로 제품전체를 분할한다. 그다음 매 모듈이 설계되는데 이 과정을 상세설계라고 부른다. 이 두개의 설계문서들은 《제품이 어떻게 일하는가?》를 서술하고 있다.

4. 실현단계(*Implementation phase*) 이 단계에서는 여러가지 구성요소들이 코드작성되고 시험된다.

5. 통합단계(*Integration phase*) 이 단계에서는 제품의 구성요소들이 전체적으로 결합되고 시험된다. 개발자들이 제품이 정확하게 기능을 수행한다고 생각하게 되면 그 제품은 의뢰자들에 의하여 시험된다(인수시험; *acceptance testing*). 제품이 의뢰자들에 의하여 승인되고 콤퓨터에 설치되게 되면 이 단계는 끝난다(통합단계는 실현단계와 병렬로 수행되여야 한다는것은 제15장에서 보게 된다.).

6. 유지정비단계(*Maintenance phase*) 제품은 개발목적으로 한 과제를 수행하는데 리용된다. 이 기간에 제품은 유지정비된다. 유지정비는 일단 의뢰자가 제품이 명세서에 부합된다고 만족을 표시한후 그 제품에 대하여 진행하는 모든 변화를 포함하게 된다(다

음의 《알고 싶은 문제》를 보시오.). 유지정비에는 명세서에 대한 변경들과 그 변경들에

알고 싶은 문제

1970년대에 쏘프트웨어제품생산은 련속적으로 수행되는 두개의 구별되는 활동 즉 개발에 뒤이은 유지정비로 이루어 진다고 간주되였다. 쏘프트웨어제품은 령상태에서 개발된 다음 의뢰자의 콤퓨터들에 설치되게 된다. 잔류오유를 수정하든가 기능을 확장한다든가 등 쏘프트웨어제품이 설치된후에 진행되는 모든 변경들이 고전적인 유지정비를 이룬다[IEEE 610.2, 1990]. 그러므로 쏘프트웨어가 고전적으로 개발되는 방식을 선개발후유지정비(*development-then-maintenance*)모형이라고 말할수 있다.

이것은 일시적인 정의에 지나지 않는다. 다시 말하여 활동은 그것이 진행되는 시점에 따라서 개발 또는 유지정비로 구별된다. 이제 쏘프트웨어가 설치된후 어느 날 오유가 발견되여 정정된다고 가정하자. 이것은 명백히 유지정비로 된다. 그러나 쏘프트웨어가 설치되기전에 우와 같은 오유가 발견되여 정정된다고 하면 이것은 고전적인 정의에 의하여 개발로 된다.

다음의 두가지 리유로 하여 이러한 모형은 오늘날에 와서 비현실적인것으로 된다. 첫째로 요즘 하나의 제품을 개발하는데는 보통 1년 또는 그이상 시간이 걸린다. 이 기간에 의뢰자들의 요구는 변화될수도 있다. 실례로 의뢰자는 쏘프트웨어제품이 리용가능한 보다 더 빠른 처리기에서 실현될것을 주장할수도 있다. 한편 제품이 개발되는 도중에 의뢰자측은 카나다로 확장되여 이제는 제품이 카나다에서도 판매를 실현할수 있도록 수정되여야 할수도 있다. 요구사항에서의 이러한 변화가 쏘프트웨어생명주기에 어떤 영향을 미치는가를 보기 위하여 설계가 진행되고 있는 동안 의뢰자들의 요구사항이 변화된다고 가정하자. 쏘프트웨어개발팀은 개발을 중지하고 변화된 요구사항을 반영하여 명세서를 수정하여야 한다. 더우기 만일 명세서에 대한 변경이 이미 완성된 설계의 부분들에 대하여 해당한 변경을 진행할것을 요구한다면 그때는 설계도 물론 수정되여야 한다. 이러한 변경이 다 진행되였을 때에만 개발을 계속 진행할수 있다. 달리 말하여 개발자들은 제품이 설치되기전에 오래동안 《유지정비》를 진행하여야 한다.

둘째로 고전적인 선개발후유지정비모형에서 쏘프트웨어를 개발하는 방법상의 문제로써 제기된다. 고전적인 쏘프트웨어공학에서 개발의 특징은 개발팀이 목적하는 제품을 령상태에서부터 개발한다는것이다. 이와 대조적으로 오늘날 쏘프트웨어생산의 비용이 높아 지는것과 관련하여 개발자들은 개발되여야 할 쏘프트웨어에서 될수록 현존쏘프트웨어제품의 일부분을 재리용하려고 하고 있다(재리용에 대하여서는 제8장에서 자세히 서술한다.). 따라서 선개발후유지정비모형은 재리용이 진행될 때에는 적합치 않다.

유지정비를 고찰하는데서 보다 더 현실적인 방도는 유지정비를 《쏘프트웨어가 개선이나 적응과 관련한 어떤 문제나 요구로 하여 코드 또는 그와 관련한 문서들에 대한 수정을 거치게 된다.》고 할 때 일어 나는 과정으로 간주하는것이다[ISO/IEC 12207,1995]. 이 정의에 따르면 제품이 설치되기전이나 설치된 다음에 진행하는가에 관계없이 오유가 수정되거나 오유가 변화될 때마다 유지정비가 진행된다.

그러나 대부분의 쏘프트웨어공학자들이 선개발후유지정비모형이 시대에 뒤떨어 진것이라는것을 깨닫게 되기전까지는 단어 유지정비(*maintence*)의 리용을 달리 하는데는 거의 문제가 없다. 이 책에서는 제품이 완성되여 설치된 다음에 진행하는 유지정비에 대하여 언급하기로 한다. 그러나 유지정비의 실제적인 본질은 곧 보다 더 광범하게 인식될것이라고 생각한다.

대한 실현으로 이루어 지는 확장(또는 쏘프트웨어갱신)은 물론 명세서를 변화시키지 않은채 잔류오유들을 제거하는 교정유지정비(또는 쏘프트웨어수정)가 포함된다.

확장에는 두가지 류형이 있다. 첫째는 완전유지정비(*perfective maintenance*) 즉 추가적인 기능 또는 감소된 응답시간과 같이 의뢰자가 제품의 효과성을 개선하기 위하여 진행하는 변경들이다. 둘째는 적응유지정비(*adaptive maintenance*) 즉 새로운 관리조절과 같이 제품이 동작하는 환경에서의 변화에 대하여 만들어 진 변경들이다. 연구결과는 유지정비성원들이 평균적으로 교정유지정비에 약 17.5%, 완전유지보에 60.5%, 적응유지정비에 18% 정도의 시간을 소비하게 된다는것을 보여 주었다.

7. 페기(*Retirement*) 이 단계에서 제품은 봉사로부터 제거된다. 페기는 그 제품이 제공하는 기능이 더이상 의뢰자측에 아무런 쓸모도 없는 경우에 발생한다.

유지정비의 론점으로 되돌아 가서 보면 때때로 나쁜 쏘프트웨어제품만이 유지정비를 거쳐야 한다고 한다. 사실상 그 의견은 옳다. 즉 나쁜 제품은 버려 지고 반면에 좋은 제품은 약 10, 15 지어 20년까지도 수리되고 기능이 확대된다. 더우기 하나의 쏘프트웨어제품은 현실세계에 대한 하나의 모형이며 현실세계는 끊임없이 변화되게 된다. 결국 쏘프트웨어는 현실세계를 정확히 반영하도록 끊임없이 유지정비되여야 한다.

실례로 판매세금률이 6%에서 7%까지 변하면 구매 또는 판매를 처리하는 거의 모든 쏘프트웨어제품들은 변경되여야 한다. 제품이 C++명령

```
const float salesTax = 6.0
```

을 포함하거나 그것과 동등한 Java명령

```
public static final float salesTax = (float) 6.0;
```

을 포함한다고 하자. 여기서 이 명령은 salesTax는 값 6.0으로 초기화된 류동소수점상수라는것을 선언하고 있다. 이 경우에 유지정비는 비교적 간단하다. 본문편집기로 값 6.0을 7.0으로 교체하고 코드를 다시 콤파일하고 련결한다. 그러나 만일 판매세금의 값이 호출될 때마다 이름 salesTax대신에 실제값 6.0이 이 제품에서 리용되였다면 그러한 제품은 유지정비하기가 아주 어렵게 된다. 실례로 원천코드에서 7.0으로 변경되여야 하지만 스쳐보낸 값 6.0이 있을수도 있고 또는 판매세금과는 관계가 없지만 부정확하게 7.0으로 변경되는 값 6.0이 있을수도 있다. 이러한 오유를 찾는것은 거의 언제나 어렵고 시간이 많이 소비된다. 사실상 어떤 쏘프트웨어에서는 많은 상수들가운데서 어느것이 변경되여야 하며 또 그 변경을 어떻게 진행하겠는가를 결정하려고 하는것보다 그 제품을 버리고 다시 코드작성하는것이 시간이 적게 걸릴수도 있다. 실시간적인 현실세계는 끊임없이 변화한다. 어떤 분사식전투기들에 장비된 미싸일들은 련관된 항공전자설비체계의 무기조종요소들을 변화시킬것을 요구하므로 새로운 모형으로 교체되게 된다. 일반적인 4기통자동차들이 6기통기관으로 교체되게 된다. 즉 이것은 연료분사식체계와 시간계수기 등을 조종하는 자동차에 장비된 콤퓨터를 변경시켜야 한다는것을 의미한다. 건전한 개발기업체들은 변경되지만 사멸하는 개발기업체들은 더이상 발전하지 않는다. 이것은 확대형식의 유지정비가 개발기업체가 살아 움직인다는것을 보여 주면서 개발기업체의 활동에 긍정적인 기여를 한다는것을 의미한다. 그러면 유지정비하는데 얼마나 많은 시간이 걸리겠는가?

그림 1-2에 있는 파라다임도표는 문헌 [Elshoff, 1976; Daly, 1977; Zelkowitz, Shaw, and Gannon, 1979; and Boehm, 1981]을 비롯한 각이한 자원들로부터 얻은 자료를 평균하여 얻어 낸것이다. 그림 1-2는 쏘프트웨어생명주기의 매 단계에서 소비되는 시간(=비용)에 대한 근사적인 퍼센트를 보여 주고 있다. 약 15년후에도 여러 개발단계들에 지출되는 시간비률은 거의 변화되지 않았다. 이것을 그림 1-3에 제시하였는데 여기서는 그림 1-2에 있는 자료를 132개 휼레트-패카드(Hewlett-Packard)프로젝트들의 최신자료와 비교하고 있다. 그림 1-2에 나오는 자료는 보다 새로운 자료와 비교할수 있도록 분할되였다.*)

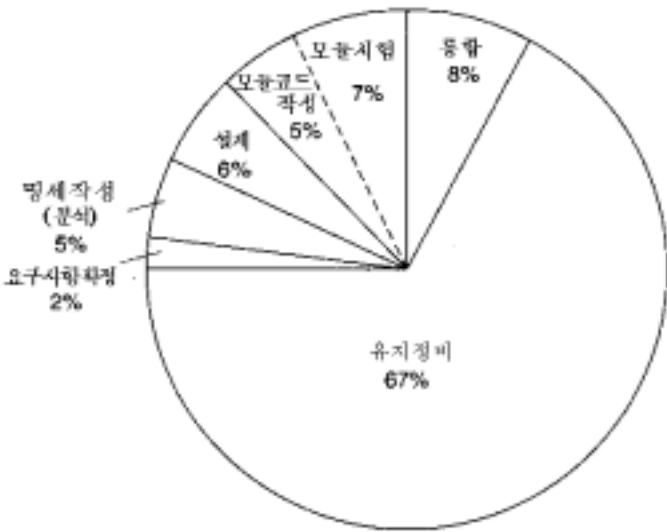



그림 1-2. 쏘프트웨어생명주기의 단계별 근사적인 상대비용

그림 1-2에 보여 준바와 같이 전체 쏘프트웨어비용의 약 2/3가 유지정비에 돌려 진다. 최근의 자료들은 유지정비의 중요성에 대하여 계속 강조하고 있다. 실례로 1990년에 휼레트-패카드회사에서의 연구 및 개발성원들중 60～80%는 유지정비에 종사하였으며 유지정비에 드는 비용은 쏘프트웨어의 전체 비용의 40～60%에 달하였다[Coleman, Ash, Lowther, and Oman, 1994]. 그러나 많은 개발기업체들에서는 시간과 노력의 80%정도를 유지정비에 돌리고 있다[Yourdon, 1996]. 그러므로 유지정비는 쏘프트웨어생명주기가운데서 매우 많은 시간을 소모하는 단계로 된다.

이제 현재 코드작성기법 CT_{old}를 리용하고 있는 쏘프트웨어기업체들에서 CT_{new}가 코드작성시간을 약 10% 감소시키게 될것이라는것을 들은 경우를 다시 고찰해 보자. 비록 CT_{new}가 유지정비에 전혀 역작용을 하지 않는다 할지라도 예민한 쏘프트웨어관리자는 코드작성실행을 바꾸기전에 다시 한번 생각할것이다. 전체 성원들은 구입한 새 쏘프트웨어 개발도구에 숙련되여야 할것이며 아마도 새 기법에 경험이 있는 성원들을 보충해야 할수

*) 그림 1-3은 다만 개발단계를 반영하고 있다. 그림 1-2에 있는 요구사항확정과 명세작성단계들에 드는 개발시간의 몫은 그림 1-3에 보여 준바와 같이 (2+5)/33 또는 21%이다.

도 있다. 이 모든 비용과 재조직은 쏘프트웨어비용에서 가능한껏 0.5%정도 감소시켜야 한다. 왜냐하면 그림 1-2에 보여 준바와 같이 모듈코드작성은 전체 쏘프트웨어비용에서 평균 5%정도만을 이루고 있기때문이다.

	1976년과 1981년사이의 프로젝트	휼레트-패카드회사의 최근 132개 프로젝트
요구사항확정단계와 명세작성(분석)단계	21%	18%
설계단계	19	18
실현단계	34	36
통합단계	29	24

그림 1-3. 1976년과 1981년사이 여러가지 프로젝트들과 휼레트-패카드회사의 최근 132개 프로젝트의 개발단계에 따르는 시간소비의 대략적인 평균퍼센트비교

이제 유지정비비용을 10%로 줄이는 새 기술이 개발되였다고 하자. 이것은 아마 즉시에 도입되게 될것이다. 왜냐하면 전체적인 비용이 평균적으로 약 6.7%정도 감소될수 있기때문이다. 이러한 기법으로 전환하는데 드는 무익한 시간은 그러한 큰 규모의 전면적인 절약에 적은 비용을 지불하는것으로써 보상된다.

유지정비가 이처럼 중요하기때문에 유지정비의 비용을 줄이는 기법이나 도구, 실천은 쏘프트웨어공학의 중요한 측면을 이루게 된다.

1. 4. 명세작성 및 설계측면

쏘프트웨어전문가들도 사람인것만큼 때때로 제품을 개발하는데서 오유를 범하게 된다. 결과 쏘프트웨어에 오유가 있게 된다. 만일 요구사항확정단계에서 오유가 생기게 되면 그로부터 초래되는 오유는 명세작성과 설계 그리고 코드에 반영되게 된다. 오유를 제때에 수정할수록 더 좋다는것은 명백하다. 쏘프트웨어생명주기의 여러단계들에서 오유를 수정하는데 드는 상대적인 비용들을 그림 1-4에 보여 주었다[Boehm, 1981]. 이 그림은 IBM[Fagan, 1974], GTE[Daly, 1977], 보호프로젝트[Stephenson, 1976] 그리고 어떤 소규모TRW프로젝트[Boehm, 1980]에서의 자료를 반영하고 있다.

그림 1-4에서 실선은 보다 큰 프로젝트와 관련되는 자료들에 가장 적합한것이고 점선은 보다 작은 프로젝트와 관련된 자료들에 가장 적합한것이다. 쏘프트웨어생명주기의 매 단계들에서 오유를 발견하고 정정하기 위한 대응하는 상대비용들을 그림 1-5에 보여 주었다. 그림 1-5에 있는 실선에서 매 계단은 그림 1-4의 실선우에서 대응하는 점들을 취하고 선형축척으로 자료들을 찍음으로써 구성되였다.

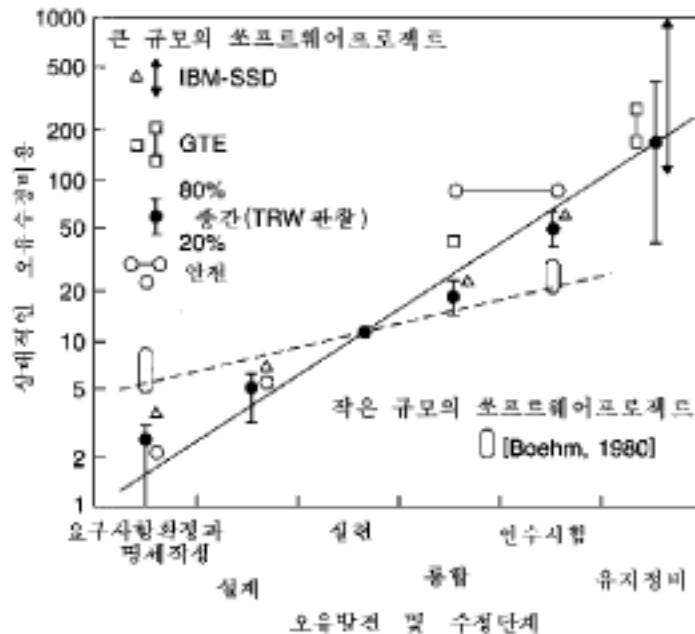



그림 1-4. 쏘프트웨어생명주기의 여러단계에서 오유를 수정하는데 드는 상대적인 비용
실선은 보다 큰 쏘프트웨어프로젝트와 관련한 자료이며 점선은
보다 작은 쏘프트웨어프로젝트와 관련한 자료이다.
(Prentice Hall, Inc., Englwood Cliffs, NJ의 허락을 받은 배리 보엠의
Software Engineering Economic, ©1981, 40페지)

설계단계에서 어떤 특정한 오유를 발견하고 정정하는데 40딸라의 비용이 들었다고 가정하자. 그림 1-5(1974년과 1980년사이에 있는 프로젝트)에 있는 실선으로부터 명세작성단계에서 그와 같은 오유를 수정하는데는 약 30딸라 소비되였다. 그러나 유지정비단계에서는 그러한 오유를 찾아 내고 수정하는데는 약 2,000딸라가 들었다. 이 자료는 오유를 빨리 찾아 내는것이 매우 중요하다는것을 보여 준다.

그림 1-5에서 점선은 IBM AS/400에 대한 체계프로그람개발과정에 제기된 오유를 발견하고 수정하는데 든 비용을 보여 준다. 평균적으로 AS/400쏘프트웨어의 유지정비단계에서는 그와 같은 오유를 수정하는데 3,680딸라의 비용이 들었다.

어떤 오유를 정정하는데 드는 비용이 그렇게 급격히 늘어 나는 리유는 바로 오유를 정정하기 위하여 무엇을 해야 하는가 하는것과 관련된다. 개발생명주기의 초기에 제품은 본질상 종이우에 존재할뿐이였고 오유를 정정하는 작업은 지우개와 연필을 가지고 간단히 진행할수 있었다. 또 다른 극단은 제품이 이미 의뢰자에게 배포된 경우이다. 적어도 오유를 정정한다는것은 코드를 편집하고 그것을 다시 콤파일하여 련결하고 그다음 문제가 해결된다는것을 주의 깊게 시험한다는것이다. 다음으로 변경을 진행하는것이 그 제품의 다른 곳에 어떤 새로운 문제를 발생시키지 않았다는것이 아주 중요한 문제이다. 그다음 지도서를 비롯한 련관된 문서들이 모두 갱신되여야 한다. 마지막으로 정정된 제품이 배포되여 설치되여야 한다.

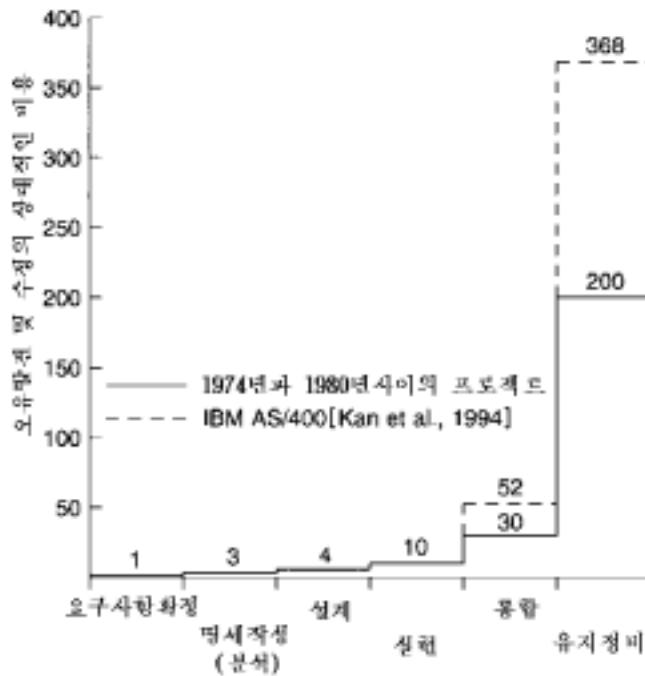



그림 1-5. 실선은 그림 1-4에 있는 선형의 실선에 있는 점들을 나타낸다.
점선은 보다 새로운 자료를 나타낸다.

이로부터 얻게 되는 교훈은 다음과 같다. 즉 오유는 될수록 빨리 찾아야 하며 오유를 정정하는데는 자금이 든다는것이다. 따라서 요구사항확정과 명세작성(분석)단계에서 오유를 발견하기 위한 기법들을 소유해야 한다. 이런 기법들은 더욱더 절실히 요구된다. 연구결과는 큰 규모의 프로젝트에서 발견된 모든 오유의 60～70%가 명세작성이나 설계상의 오유였다는것을 보여 주었다. 보다 새로운 결과는 명세작성과 설계상의 오유가 더 크다는것을 확증해 주었다. 검토는 어떤 팀이 작성한 문서에 대한 주의 깊은 시험이다(6.2.3). 태양계의 우주공간프로그람을 무력하게 만드는 NASA의 분사식추진기연구쏘프트웨어에 대하여 203개의 검토를 진행하는 기간에 평균 페지당 오유가 명세서에서는 약 1.9개, 설계에서는 0.9개, 코드에서는 0.3개가 발견되였다[Kelly, Sherif, and Hops, 1992]. 따라서 명세작성 및 설계기법을 개선하는것이 중요하다. 그리하여 될수록 빨리 오유를 발견하여야 한다. 왜냐하면 명세작성과 설계에서의 오유가 모든 오유들가운데서 큰 몫을 차지하기때문이다. 유지정비비용을 10% 감소시키면 총체적인 비용은 거의 7% 감소될것이라는것을 보여 준 앞부분의 설계에서와 마찬가지로 명세작성과 설계오유를 10% 감소시키면 오유의 총 개수가 6～7% 감소될것이다.

여러가지 프로젝트들에 대한 새로운 자료들이 문헌 [Bhandari et al., 1994]에 서술되였다. 실례로 콤파일러는 방대한 변화를 거치였다. 설계단계의 마감에서 프로젝트개발기간에 발견된 오유는 분류되였다. 13%의 오유만이 이전의 판본의 콤파일러들에서부터 이

전되여 왔다. 나머지오유가운데서 16%는 명세작성단계에서, 71%는 설계단계에서 인입되였다. 쏘프트웨어생명주기의 초기에 그렇게 많은 오유들이 인입된것은 쏘프트웨어공학의 또 하나의 중요한 측면을 강조하고 있다. 즉 보다 좋은 명세와 설계를 생성하는 기법들을 강조하고 있다.

대부분의 쏘프트웨어는 개발 및 유지정비단계를 책임진 개별적인 사람들이 아니라 쏘프트웨어공학개발팀들에 의하여 개발되게 된다. 이제 이 의미를 고찰해 보자.

1. 5. 개발팀프로그람작성측면

어떤 콤퓨터의 성능 - 비용인자는 다음과 같이 정의될수 있다.

성능 - 비용인자 = 백만개의 더하기를 수행하는 시간 × CPU와 주기억의 가격

이 값은 새세대 콤퓨터들의 출현과 함께 감소되였다. 이러한 감소는 전자공학 특히는 3극소자와 대규모집적회로(VLSI)기술이 발견된 결과에 이루어 진것이다. 이러한 발견결과로 개발기업체들은 큰 제품들 즉 허용된 시간제약내에 한사람이 작성하기에는 너무 큰 제품들을 동작시킬수 있는 하드웨어를 제공할수 있었다. 실례로 어떤 제품이 18개월이내에 배포되여야 하는데 한명의 프로그람작성자가 그것을 15년동안 완성해야 한다면 그러한 제품은 개발팀에서 개발하여야 한다. 그러나 개발팀프로그람작성은 코드구성요소들사이의 결합문제와 개발팀성원들사이의 통신문제를 야기시킨다.

실례로 죠(Joe)와 프레다(Freda)가 각각 모듈 p와 q를 코드작성하는데 여기서 모듈 p는 모듈 q를 호출한다. 모듈 p를 작성할 때 죠는 인수목록에 5개의 인수를 가지고 q에 대한 호출을 하게 된다. 프레다는 5개의 변수를 가지고 코드 q를 작성하였지만 죠가 한것과는 다른순서로 하였다. 함수원형을 리용하지 않으면 이것은 ANSI C콤파일러에서 발견되지 않을것이다. Java해석기와 적재프로그람, C에서의 *lint*(8.7.4) 또는 Ada의 련결프로그람과 같은 일부 쏘프트웨어도구들은 오직 호상교환변수들의 형이 다를 때에만 이러한 형위반을 발견할수 있다. 즉 그것들의 형이 같으면 오랜 기간 문제가 발견되지 않는다. 이것은 설계상의 문제이라고 론의되였으며 만일 모듈이 주의 깊게 설계되면 이 문제는 제기되지 않는다. 이것은 사실이나 실천적으로 설계는 흔히 코드작성을 개시한 다음에 변화되는데 그러한 변화에 대한 통보는 개발팀의 모든 성원들에게 배포되지 않을수도 있다. 이리하여 둘이상의 프로그람작성자들이 작업하는 설계가 변화될 때 불충분한 통신은 죠와 프레다가 체험한 결합문제들을 초래할수 있다. 큰 제품을 실행할수 있는 강력한 콤퓨터들이 제공될수 있게 되기전과 마찬가지로 다만 한명의 개발성원이 제품의 모든 측면을 책임지게 되는 경우에는 이러한 문제가 덜 발생할수도 있다.

그러나 결합문제는 개발팀이 쏘프트웨어를 개발할 때 생겨 날수 있는 문제들에 비하면 빙산의 일각에 지나지 않는다. 개발팀이 합리적으로 조직되지 않으면 개발팀성원들사이의 협력에 지나치게 많은 시간이 랑비될수 있다. 제품이 단 한명의 프로그람작성자에 의하여 1년동안에 완성된다고 가정하자. 만일 같은 과제를 3명의 프로그람작성자들로 조직된 개발팀에게 맡긴다면 그 과제를 완성하는데 4개월이 아니라 1년까지 걸릴수 있으며 결과

코드의 품질은 전체적인 과제를 한명에게 맡기는것보다 더 떨어 지게 될것이다. 오늘날 쏘프트웨어의 많은 부분은 개발팀에 의하여 개발되고 유지정비되기때문에 쏘프트웨어공학의 범위는 개발팀이 합리적으로 조직되고 관리될것을 담보하는 기법들을 포함하여야 한다.

앞절들에서 보여 준바와 같이 쏘프트웨어공학의 범위는 아주 넓다. 그것은 요구사항확정단계로부터 시작하여 페기단계에 이르기까지 쏘프트웨어생명주기의 모든 단계들을 포함하게 된다. 그것은 또한 개발팀과 같은 인간적인 측면들과 경제적인 측면, 저작권법과 같은 법률상의 측면을 포함하게 된다. 이 모든 측면들은 이 장의 첫 부분에서 준 쏘프트웨어공학의 정의에서 명백히 구체화되였다. 즉 쏘프트웨어공학은 제때에 주어 진 예산내에서 사용자들의 수요를 만족시키면서 배포되는 오유 없는 쏘프트웨어제품을 개발하는것을 목적으로 하는 학문이다.

1. 6. 객체지향파라다임

1975년이전에는 대부분의 쏘프트웨어기업체들이 특정한 기법을 리용하지 않았다. 즉 개별적으로 자기식대로 연구를 진행하였다. 기본돌파구는 1975년부터 1985년사이에 이른바 구조화파라다임이 개발되면서 이루어 졌다. 구조화된 파라다임을 구성하는 기법에는 구조화체계분석(11.3)과 자료흐름분석(7.1), 구조화프로그람작성과 구조화시험(14.8.2)들이 포함된다. 이러한 기법들이 처음 리용될 때는 아주 전망적인것 같았다. 그러나 시간이 지남에 따라 성공적이지 못하다는것이 두가지 측면에서 증명되였다. 그것은 첫째로 그 기법들은 때때로 쏘프트웨어제품의 크기가 증대됨에 따라 그에 대처할수 없다는것이다. 즉 구조화기법은 5,000 또는 50,000행이나 되는 코드를 가진 제품을 취급할 때는 충분하였다. 그러나 오늘 50만행이나 되는 코드를 포함하는 제품들은 크다고 간주되지 않는다. 지어 500만행이상되는 코드를 가진 제품도 드물지 않다. 구조화기술은 흔히 오늘날의 큰 제품들을 관리할수 있을 정도로 확장되지 않을수도 있다.

유지정비단계는 구조화파라다임을 더이상 기대할수 없도록 하는 두번째 령역으로 된다. 30여년전 구조화파라다임을 개발하여야 하였던것은 평균쏘프트웨어예산의 2/3가 유지정비에 돌려 지고 있었기때문이다(그림 1−2). 그런데 구조화파라다임은 이러한 문제를 해결할수 없었다. 즉 1.3에서 지적한바와 같이 많은 개발기업체들은 여전히 유지정비에 80%의 시간과 노력을 돌리고 있다[Yourdon, 1996].

구조화파라다임이 크게 성공하지 못한 리유는 구조화기법이 행동지향적기법이나 또는 자료지향적기법으로는 되지만 두가지 기법을 모두 지향하지는 않기때문이다. 쏘프트웨어제품의 기본구성요소는 제품의 작용*)들과 그 작용들이 조작하는 자료들이다.

실례로 평균높이를 결정한다(determine average height)는 높이(자료)들의 모임에 대하여 조작을 진행하고 그 높이(자료)들의 평균값을 귀환하는 작용이다. 자료흐름분석(13.3)과 같은 일부 구조화기법들은 작용지향적이다. 즉 이러한 기법들은 제품의 작용들에 주

---

*) 이 책에서는 쏘프트웨어개발공정(*software process*)이라는 용어와 혼돈을 피하기 위하여 오히려 작용(*action*)이라는 단어를 리용한다.

목하고 있다. 즉 자료의 중요성은 2차적인것으로 된다. 반대로 잭슨(Jackson)체계개발(13.5)과 같은 기법들은 자료지향적이다. 여기서는 자료에 중점을 둔다. 즉 자료에 대한 작용에는 큰 의의를 부여하지 않는다.

이와 대조적으로 객체지향파라다임에서는 자료와 작용을 다같이 중요하게 보고 있다. 어떤 객체를 고찰하는 가장 단순한 방법은 객체를 자료와 그 자료에 대한 작용을 모두 병합하고 있는 하나의 통합된 쏘프트웨어구성요소로 보는것이다. 이러한 정의는 완전하지 않은데 이 책의 뒤부분에서 일단 계승(*inheritance*)이 정의되면(7.7) 다시 정의될것이다. 그러나 이 정의는 객체의 본질에 대하여 많은것을 말해 주고 있다. 객체에 대한 한가지 실례로 은행업무를 들수 있다(그림 1-6). 이 객체의 자료요소는 회계맞추기(*account balance*)이다. 회계맞추기에 대하여 수행할수 있는 작용들은 예금(*deposit*), 반환금(*withdraw*)과 잔고결정(*determine balance*)들이다. 구조화파라다임의 관점에서 보면 은행업무를 취급하는 제품은 하나의 자료요소 회계맞추기와 세개의 작용 예금, 반환, 잔고결정을 통합하여야 한다. 객체지향관점에서 보면 은행업무는 하나의 객체로 된다.

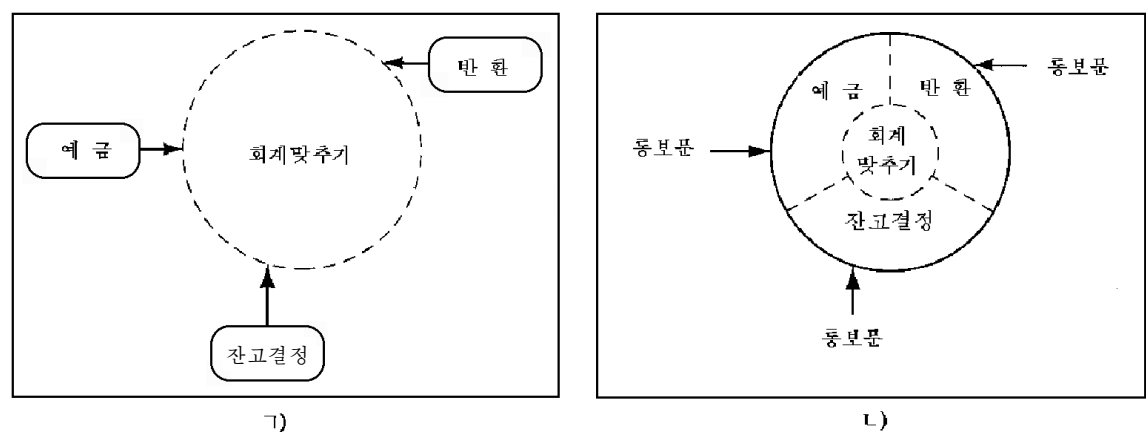



그림 1-6. 구조화파라다임(ㄱ)과 객체지향파라다임(ㄴ)을 리용한 은행업무실현에 대한 비교
객체를 둘러 싼 검은 실선은 회계맞추기를 어떻게 실현하는가에 대한 세부에
대하여서는 객체밖에서는 알수 없다는것을 보여 준다.

이 객체는 자료요소와 그 자료요소에 대하여 수행되는 세개의 작용들을 하나의 단위로 결합하고 있다.

지금까지는 이 두가지 방법사이에 약간 차이가 있는것 같다. 그러나 중요한 점은 어떤 객체가 실현하는 방식에 있다. 특히 어떤 객체의 자료요소가 어떻게 기억되는가에 관한 세부들은 그 객체의 외부에는 알려 지지 않는다. 이것이 《정보은페(*information hiding*)》의 한가지 실례인데 이것에 대하여서는 7.6에서 더 자세히 론의한다. 그림 1-6의 ㄴ)에서 보여 준 은행업무객체의 경우에 이 쏘프트웨어제품의 나머지부분은 은행업무객체내에서의 잔고와 같은것이라는것을 알수 있지만 회계맞추기의 형식에 관해서는 그 어떤 개념도 가질수 없다. 즉 이 객체의 밖에서는 회계맞추기가 옹근수로 실현되는가 또는 류동소수점

으로 실현되는가를 전혀 알수 없다. 객체를 둘러 싼 이러한 정보장벽은 그림 1-6의 ㄴ)에서 검은 실선으로 표시되는데 그것은 객체지향파라다임을 리용한 하나의 실현에 대하여 설명하고 있다. 반대로 그림 1-6의 ㄱ)에서 회계맞추기는 점선으로 둘러 싸여 진다. 왜냐하면 회계맞추기의 모든 세부들이 구조화파라다임을 리용한 실현에서 모듈로 고찰되며 회계맞추기의 값이 그중 어느 한 모듈에 의하여 변화될수 있기때문이다.

그림 1-6의 ㄴ)의 객체지향실현을 고찰하면 만일 손님이 하나의 계산서에 10딸라를 예금하면 회계맞추기자료요소(속성; *attribute*)에 10딸라를 불쿠어 주라는 하나의 통보(*message*)가 련관된 객체의 예금작용(방법(*method*))에 보내진다. 예금(*deposit*)방법은 은행업무객체내에 있으며 회계맞추기가 어떻게 실현되는가를 알고 있다. 이것은 객체안에서 점선으로 지적되고 있다. 그러나 이 객체밖의 그 어떤 실체도 이러한 지식을 가지고 있을 필요가 없다.

그림 1-6의 ㄴ)에 있는 세개의 방법이 회계맞추기를 제품의 나머지부분으로부터 차단하고 있는것은 이러한 지식의 국부화를 보여 주고 있다.

얼핏 보기에는 실현세부들이 어떤 객체에 국부적이라는 사실이 그리 쓸모 있는것 같지는 않을수도 있다. 우선 은행업무에 관한 제품이 구조와 파라다임을 리용하여 구축된다고 하자. 만일 회계맞추기를 표시하는 방법이 하나의 옹근수로부터 하나의 구조체마당으로 변화되게 되면 회계맞추기와 함께 진행되는 제품의 모든 부분이 변화되게 되며 이때 이러한 변화들은 모순이 없이 진행되여야 한다. 반대로 객체지향파라다임이 리용된다면 다만 은행업무객체 그자체내에서만 변화가 진행되게 될것이다. 제품의 다른 부분들은 회계맞추기가 어떻게 실현되는지 알지 못하며 따라서 기타 다른 부분들은 회계맞추기를 호출할수 없다. 따라서 은행업무제품의 기타 다른 부분들은 변경될 필요가 없다. 결국 객체지향파라다임은 유지정비를 보다 빨리 보다 쉽게 할수 있게 하며 회귀오유(즉 제품의 다른 부분에 명백히 상관이 없는 변경을 진행한 결과에 그 제품의 어느 한 부분에 본의아니게 인입된 오유)의 발생은 크게 줄어 들게 될것이다.

유지정비의 우월성외에도 객체지향파라다임은 개발을 보다 쉽게 할수 있게 한다. 많은 실례들에서 하나의 객체는 어떤 물리적인 등가물을 가진다. 실례로 은행업무제품에서 대상 은행업무(bank account)는 은행에서의 실제적인 은행업무에 대응한다. 제12장에서 보여 주는바와 같이 모형화는 객체지향파라다임에서 큰 역할을 논다. 제품에서의 객체와 현실세계에서의 그 대응물사이의 밀접한 대응은 더 좋은 쏘프트웨어를 개발할수 있게 한다.

객체지향파라다임의 유익성을 또 다른 측면에서 찾아 볼수 있다. 잘 설계된 객체는 독립적인 구성단위들이다. 이미 설명된바와 같이 객체는 자료와 그 자료에 대하여 수행되게 되는 작용들로 구성되여 있다. 만일 객체의 자료에 대하여 수행하게 되는 모든 작용이 그 객체에 포함되면 객체를 개념상 독립적인 실체로 고찰할수 있다. 객체로 모형화되는 현실세계의 부분들과 관련되는 제품안의 모든 부분들은 객체 그자체에서 찾아 볼수 있다. 이와 같은 개념적독립성을 때때로 교갑화(*encapsulation*)라고 부른다(7.4). 그러나 물리적독립성이라고 하는 추가적인 형태의 독립성이 존재한다. 잘 설계된 어떤 객체에서는 정보은페가 실현세부들이 그 객체밖의 모든것으로부터 은페된다는것을 담보하여 준다. 유일하게 허용되는 통신형식은 어떤 특정한 작용을 수행하기 위하여 그 객체에 통보를 보내는것이다. 작용이 수행되는 방법은 전적으로 객체 그자체의 책임이다. 이러한 리유로 하여 객체

지향설계를 때때로 책임구동설계(*responsibility-driven des i gn*)[Wirfs-Brock, Wilkerson, and Wiener, 1990] 또는 계약에 의한 설계(*design by contract*)[Meyer, 1992a]라고 부른다(책임구동설계에 대한 다른 견해에 대하여서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

당신이 뉴 오를레안(New Orleans)에서 살고 있는데 아이오와(Iowa)시에서 살고 있는 작은 어머니의 생일에 선물할 꽃을 마련하려 한다고 하자[Budd, 1991]. 방법은 아이오와시에 있는 모든 화초재배목록을 찾고 그다음 작은어머니집에 가장 가까이에 있는 화초재배원을 결정하는것이다. 보다 쉬운 방법은 1-800-FLOWERS에 전화를 걸어 그 단위에 꽃선물을 전달할것을 위임하는것이다. 당신은 꽃을 전달할 아이오와시의 화초재배가를 알 필요가 없다.

이와 같은 방식으로 어떤 통보가 하나의 객체에 전송될 때 그 요청이 어떻게 수행되는가는 전혀 관계가 없을뿐아니라 지어 그 통보문을 보내는 단위가 그 객체의 내부적인 구조에 대하여 아는것을 허락하지 않는다. 객체 그자체들은 그 통보문을 수행하는 매 세부를 전적으로 책임진다.

구조화파라다임을 리용하여 개발한 제품은 본질에 있어서 하나의 독립적인 단위이다. 이것은 구조화파라다임이 보다 큰 제품들에 응용될 때 그리 성공적이지 못한 하나의 원인으로 된다. 이와 반대로 객체지향파라다임이 정확히 리용되면 그 결과 개발된 제품은 여러개의 보다 작고 독립성이 큰 단위들로 구성된다. 객체지향파라다임은 쏘프트웨어제품의 복잡도를 줄이며 이로부터 개발과 유지정비가 간단해 진다. 객체지향파라다임의 또 한가지 긍정적인 특징은 그것이 재리용을 촉진시킨다는것이다. 즉 객체는 독립적인 실체이기때문에 미래의 제품들에서 리용될수 있다. 객체들에 대한 이러한 재리용은 제8장에서 설명한바와 같이 개발과 유지정비에서 시간과 비용을 감소시킨다.

객체지향파라다임이 리용되면 쏘프트웨어생명주기(그림 1−1)는 약간 변경되여야 한다. 그림 1−7은 구조화파라다임과 객체지향파라다임에 대한 두개의 쏘프트웨어생명주기를 보여 준다. 그 차이를 명백히 하기 위하여 우선 구조화파라다임의 설계단계를 고찰하자. 1.3에서 서술한바와 같이 이 단계는 두개의 부분적인 단계 즉 구성방식설계와 그다음의 상세설계로 나뉘여 진다. 구성방식설계단계에서 제품은 모듈(*module*)이라고 부르는 구성요소들로 분해된다. 상세설계단계에서 매 모듈의 자료구조와 알고리듬들이 차례로 설계된다. 마지막으로 실현단계에서 이러한 모듈들이 실현된다.

만일 객체지향파라다임이 대신 리용된다면 객체지향분석단계의 한 단계는 객체를 결정한것이다.

객체는 일종의 모듈이기때문에 구성방식설계는 객체지향분석단계에서 수행된다. 이리하여 객체지향분석은 구조화파라다임의 대응하는 명세작성(분석)단계보다 먼저 진행된다. 이것을 그림 1−8에 보여 준다.

1.	요구사항확정단계	1.	요구사항확정단계
2.	명세작성(분석)단계	2'.	객체지향분석단계
3.	설계단계	3'.	객체지향설계단계
4.	실현단계	4'.	객체지향프로그람작성단계
5.	통합단계	5.	통합단계
6.	유지정비단계	6.	유지정비단계
7.	페기	7.	페기

그림 1-7. 구조화파라다임과 객체지향파라다임의 생명주기의 비교

두 파라다임사이의 이와 같은 차이로부터 중요한 결론을 얻는다. 구조화파라다임이 리용될 때는 거의 언제나 분석(명세작성)단계와 설계단계사이에 예민한 변이가 존재한다. 결국 명세작성단계의 목적은 제품이 무엇을 하는가 하는것을 결정하는것이며 반면에 설계단계의 목적은 그것을 어떻게 하여야 하는가를 결정하는것이다. 이와 대조적으로 객체지향분석이 리용될 때는 객체들은 맨 초기부터 생명주기에 들어 간다. 분석단계에서 객체들이 추출되며 설계단계에서 그것들이 설계되고 실현단계에서 코드작성된다. 결국 객체지향파라다임은 하나로 통합된 방법이다. 즉 단계와 단계사이의 이행은 구조화파라다임에서보다 훨씬 완만하며 이로부터 개발기간에 오유의 수는 감소되게 된다.

2. 명세작성(분석)단계	2'. 객체지향분석단계
・제품이 무엇을 할것인가를 결정	・제품이 무엇을 할것인가를 결정
	・객체를 추출
3. 설계단계	3'. 객체지향설계단계
・구성방식설계(모듈을 추출)	・상세설계
・상세설계	
4. 실현단계	4'. 객체지향프로그람작성단계
・적당한 프로그람작성언어로 실현	・적당한 객체지향프로그람작성언어로 실현

그림 1-8. 구조화파라다임과 객체지향파라다임의 차이

이미 언급하였지만 하나의 객체를 단순히 자료와 작용들을 모두 교갑화하고 정보은페원리를 실현하는 쏘프트웨어의 구성요소로서 정의하는것은 불충분하다. 보다 더 완성된 정의는 제7장에서 주고 있는데 거기에 객체들에 대하여 깊이 있게 서술한다. 그러나 우선 이 책에서 리용한 용어들을 보다 상세히 고찰하여야 한다.

1. 7. 용 어

이 책의 거의 모든 페지들에서 쏘프트웨어(*software*)라는 단어가 리용되고 있다. 쏘프트웨어는 기계가독형식의 코드뿐아니라 매 프로젝트의 본질적인 요소인 모든 문서들로서 이루어 진다. 쏘프트웨어는 명세서, 설계문서, 모든 종류의 법적 및 계산문서, 쏘프트웨어 프로젝트관리계획, 모든 형태의 안내서와 같은 관리문서들을 포함하고 있다.

1970년대부터 프로그람과 체계사이의 차이는 희미해 지게 되였다. 《좋은 옛시절(good old days)》에서 구별은 명백하였다. 프로그람은 일반적으로 구멍이 뚫린 착공카드형식의 실행할수 있는 자률적인 코드쪼각이였다. 체계는 련관된 프로그람들의 모임이였다. 결국 하나의 체계는 프로그람 P, Q, R, S로 구성될수 있다. 자기테프 T1이 설치되고 그다음 프로그람 P가 실행된다. 자료카드가 읽어 지게 되고 출력테프 T2, T3이 생성된다. 그다음 테프 T2가 다시 감기고 프로그람 Q가 실행되여 결국 테프 T4가 출력된다.

프로그람 R는 이제 테프 T3과 T4를 T5로 병합하며 T5는 일련의 보고서들을 인쇄하는 프로그람 S의 입력으로 된다.

앞단통신처리기와 뒤단자료기지관리기를 가진 콤퓨터상에서 실행되는 강철공장의 실시간조종을 실행하는 어떤 제품의 정황들을 비교하여 보자. 강철공장을 조종하는 하나의 토막으로 구성된 쏘프트웨어는 낡은 류형의 체계에서보다 훨씬 더잘 동작하지만 프로그람과 체계에 대한 고전적인 정의에 따르면 이러한 쏘프트웨어는 의심할바없이 프로그람이다. 체계(*system*)라는 용어는 이제는 하드웨어와 쏘프트웨어의 결합을 지적하기 위해서도 리용되고 있다. 실례로 비행기에서 비행조종체계는 비행을 위한 콤퓨터와 거기서 실행되는 쏘프트웨어로 구성되게 된다. 누가 비행조종체계라는 용어를 리용하는가에 따라서 비행조종체계는 콤퓨터와 그 콤퓨터에 의하여 조종되는 날개와 같은 부분들에 지령을 보내는 위치조종장치와 같은 그러한 조종들도 포함할수 있다.

혼란을 피하기 위하여 이 책에서는 제품이라는 용어를 쏘프트웨어의 중요한 부분을 의미하는것으로 리용한다. 이러한 약속에는 두가지 원인이 있다. 첫째는 단순히 제품이라는 용어를 리용함으로써 프로그람과 체계사이의 혼란을 없애는것이다. 둘째 리유가 더 중요하다. 이 책에서는 쏘프트웨어생산공정(*process*)을 취급하고 있는데 어떤 공정의 마지막 결과를 제품(*product*)이라고 부르고 있다. 쏘프트웨어생산은 두개의 활동 즉 쏘프트웨어개발(*software development*)과 그에 뒤따르는 유지정비(*maintenance*)로 이루어 진다. 결국 체계라는 용어는 현대적인 의미로 리용되고 있다. 즉 하드웨어와 쏘프트웨어의 결합, 또는 조작체계와 관리정보체계와 같은 일반적으로 리용되는 어휘로서 리해되고 있다.

쏘프트웨어공학에서 널리 리용되는 단어들중에는 방법론(*methodology*)과 파라다임(*paradigm*)이라는 단어가 있다. 이 두 단어는 같은 의미 즉 완전한 생명주기를 수행하기 위한 기법들의 집합으로서 리용되고 있다. 이러한 리용은 언어를 정확히 리용하는 사람들에게 혼란을 가져 온다. 즉 방법론은 방법의 과학을 의미하고 파라다임은 하나의 모형 또는 패턴을 의미한다. 쏘프트웨어공학자들이 단어들을 정확히 리용할것을 권고하고 있지만 불구하고 현실적으로는 너무나 광범히 리용되고 있는것으로하여 이 책에서는 이 두 단어를 모두 기법들의 집합이라는 의미로 리용한다.

될수록 피해야 할 하나의 용어는 바그(*bug*)이다(이 단어의 유래에 대하여서는 다음의

《알고 싶은 문제》를 보시오.).

> ## 알고 싶은 문제
>
> 오유를 의미하는 단어 바그를 처음으로 리용한 사람은 COBOL설계자의 한 사람인 이전 해군소장 그레이스 머레이 호퍼(Grace Murray Hopper)이다.
>
> 1945년 9월 9일 하바드에서 호퍼와 그의 동료들이 리용하던 MARK II 콤퓨터에 부나비 한 마리가 날아 들어 와 계전기의 접촉판사이에 끼워 들었다. 이리하여 체계에서는 실제적으로 오유가 발생되였다. 호퍼는 오유를 일지에 기록하고 《처음으로 바그의 실제적인 경우가 발견되였다.》라고 썼다. 아직도 부나비가 붙어 있는 이 일지는 버지니아주 다흘그렌(Dahlgren)에 있는 해군지상무기쎈터에 있는 해군박물관에 전시되여 있다.
>
> 이것이 콤퓨터분야에서 처음으로 리용된 바그라는 말일수도 있지만 이 단어는 19세기에 공학용어들에서 리용되였다[Shapiro, 1994]. 실례로 토마스 엘바 에디슨은 1878년 11월 18일에 다음과 같이 썼다.
>
> 《이 일이 벌어 지고 약간의 고장과 난점들과 같은 바그들이 제기될 때…》[Josephon 1992].
> 1934년판 웨브스타의 신영어사전에서는 바그에 대하여 《장치나 그 조작에서 결함》이라고 정의하고 있다.
>
> 이것이 호퍼가 그 단어를 리용한 의미와 류사하다는 호퍼의 지적으로부터 명백하다. 다시 말하여 호퍼는 자기가 의미하는바를 설명한것이다.

오늘날 용어 바그(*bug*)는 오유(*error*)를 단순히 에둘러 말한것이다. 비록 에두름법을 리용할 때 일반적으로 실제적인 손해는 없다고 할지라도 단어 바그는 좋은 쏘프트웨어생산에 기여하지 못할수도 있다. 특히 어떤 프로그람작성자는 《나는 오유를 범했다.》라고 말하지 않고 《바그가 코드에 끼워 들어 갔다.》고 말할것이다. 그러면 오유에 대한 책임이 프로그람작성자로부터 바그에로 옮겨 가게 된다. 돌림감기는 돌림감기비루스에 의하여 발생하기때문에 그 누구도 돌림감기에 걸렸다고 하여 프로그람작성자들을 꾸짖지는 않는다. 오유를 바그로 간주하는것은 책임을 회피하는 하나의 방법으로 된다. 이와 반대로 《나는 오유를 범했다.》라고 말하는 프로그람작성자는 자기의 행동에 대하여 책임을 지는 콤퓨터전문가로 된다.

객체지향에 대한 용어와 관련하여 상당히 혼돈될수 있다. 실례로 어떤 객체의 하나의 자료구성요소를 위하여 속성(*attribute*)이라는 용어외에도 때때로 상태변수(*state variable*)라는 용어가 객체지향과 관련한 문헌들에서 리용되고 있다. Java에서는 이 용어가 실례변수(*instance variable*)이며 C++에서는 이 용어가 마당(*field*)으로 쓰인다. 어떤 객체의 작용들과 관련하여 방법(*method*)이라는 용어가 흔히 리용되는데 그러나 C++에서 이 용어는 성원함수(*member function*)를 나타낸다. C++에서 어떤 객체의 성원은 속성(《마당》)이나 방법을 가리키고 있다. Java에서는 용어마당이 속성(실례변수)이나 방법을 지적하기 위하여 리용된다. 혼돈을 피하기 위하여 이 책에서는 될수록 일반적인 용어속성과 방법을 리용한다.

그러나 어떤 용어는 널리 리용되고 있다. 실례로 어떤 객체안에서 하나의 방법이 호출

되면 이것을 거의 일반적으로 그 객체에 하나의 통보문을 보낸다고 말한다.

이 절에서는 이 책에서 리용되는 여러가지 용어들을 정의하였다. 이러한 용어들중에서 공정(*process*)이라는 용어는 다음장의 주제로 된다.

요 약

쏘프트웨어공학은 주어 진 예산내에서 제때에 사용자들의 요구를 만족시키는 오유없는 쏘프트웨어제품을 개발하여 배포하는 학문으로서 정의된다(1.1). 이 목적을 달성하기 위하여 명세작성(분석), 설계(1.4), 유지정비(1.3)를 비롯한 쏘프트웨어개발의 모든 단계들에서 적절한 기법들을 리용하여야 한다. 쏘프트웨어공학은 쏘프트웨어생명주기의 모든 단계들을 취급하고 있으며 경제학(1.2)과 사회과학(1.5)을 비롯하여 여러 분야의 지식을 포함하고 있다. 1.6에서 객체에 대하여 소개하고 구조화파라다임과 객체지향파라다임을 간단히 비교하였다. 마지막부분(1.7)에서 이 책에 리용된 용어들에 대하여 설명하였다.

보 충

쏘프트웨어공학의 범위에 대한 고전적인 정보원천은 문헌 [Boehm.1976]에 있다.

문헌 [DeMarco and Lister, 1989]는 쏘프트웨어공학이 실제적으로 리용되고 있는 범위에 대하여 서술하고 있다. 쏘프트웨어공학이 진정한 공학학문으로 된다고 고찰한 분석은 문헌 [Wasserman, 1996]과 [Ebert, Matsubara, Pezzé, and Bertelsen, 1997]에 있다.

쏘프트웨어공학의 전망에 대하여서는 문헌 [Lewis, 1996a, 1996b; Leveson, 1997; Brereton et al., 1999; Kroeker et al., 1999; and Finkelstein, 2000]들에서 서술하였다. 쏘프트웨어공학위기의 요인에 대하여서는 *IEEE Software* 1999년 5월/6월호와 특히 문헌 [Reel, 1999]에서 고찰하였다.

쏘프트웨어공학의 현재 실천에 대하여서는 문헌 [Yourdon, 1996]에서 서술하고 있다. 쏘프트웨어공학에서 유지정비의 중요성에 대한 견해와 그것을 어떻게 계획하겠는가 하는 것은 문헌 [Parnas, 1994]를 참고할수 있다. 쏘프트웨어에 대한 불신임과 초래되는 위험(특히 안전성위험체계)은 문헌 [Littlewood and Strigini, 1992; Mellor, 1994; and Neumann, 1995]에서 서술하였다. 쏘프트웨어의 위기에 대한 최근의 견해는 문헌 [Gibbs, 1994]와 [Glass, 1998]에서 주었다. [Zvegintzov, 1998]에서는 쏘프트웨어공학실천에서 얼마만큼 정확한 자료들을 실제로 리용할수 있는가 하는것을 설명하였다.

수학이 쏘프트웨어공학을 안받침해 주고 있다는것은 문헌 [Parnas, 1990]에서 고찰하였다. 쏘프트웨어공학에서 경제학의 중요성에 대하여서는 문헌 [Boehm, 1981]과 [Baetjer, 1996]에서 론의하였다.

사회과학과 쏘프트웨어공학에 대한 두개의 고전적인 저서로는 문헌 [Weinberg, 1971]과 [Shneiderman, 1980]을 들수 있다. 책에서는 심리학이나 행동과학에 대한 지식을 요구하지는 않는다. 이 문제에 대한 보다 새로운 저서는 문헌 [DeMarco and Lister, 1987]이다.

브로크의 저서 신화적인 사람-달(The Mythical Man-Month)[Brooks, 1975]은 쏘프트웨어공학의 실현에 대하여 아주 상세히 소개하였다. 이 책에는 이 장에서 언급한 주제들을 모두 포함하고 있다.

객체지향파라다임에 대한 입문서적들은 문헌 [Budd, 1991]과 [Meyer, 1997]이다. 파라다임에 대하여 조리 있게 서술한 문헌은 [Radin, 1996]이다. [Khan, Al-A'ali, and Girgis, 1995]에서는 고전적인 파라다임과 객체지향파라다임사이의 차이를 설명하고 있다. 객체지향파라다임을 리용하여 성공한 프로젝트들에 대하여서는 문헌 [Capper, Colgate, Hunter and James, 1994]에서 서술하고 있다. 객체지향파라다임에 경험 있는 150여명의 쏘프트웨어개발자들에 대한 견해를 조사한 문헌은 [Johnson, 2000]이다. 대규모적인 객체지향제품을 개발하는데서 배워야 할 학과목교재로서는 [Maring, 1996]과 [Fichman and Kemerer, 1997]을 들수 있다. 문헌 [Scholtz et al., 1993]은 객체지향프로그람작성의 첨단기술과 실천에 대하여 1993년 4월에 진행한 토론회보고이다. 객체지향파라다임의 최근 동향에 대한 여러가지 간단한 기사들은 문헌 [El-Rewini et al., 1995]에서 찾아 볼수 있다. 객체지향파라다임에 대한 중요한 기사들은 *IEEE Computer* 1992년 10월호와 *IEEE Software* 1993년 1월호 그리고 *Journal of Systems and Software* 1993년 1월호와 1993년 11월호에서 볼수 있다. 객체지향파라다임의 잠재적인 함정에 관한 문제는 문헌 [Webster, 1995]에서 서술하였다.

문 제

1.1. 당신은 수자식전화기제작자들을 위하여 기본조종체계를 개발할데 대한 임무를 받았다. 개발예산은 43만딸라이다. 생명주기의 매 단계에서 얼마만한 비용이 들겠는가? 앞으로 유지정비에는 비용이 얼마나 들겠는가?

1.2. 당신은 쏘프트웨어공학고문이다. 출판업계의 부지배인이 당신에게 회사의 모든 계산업무를 수행할수 있고 여러 회사창고안에서의 주문품과 재고품목록과 관련하여 본사의 사무원에게 정보를 직결식방법으로 제공할수 있는 제품을 개발해 줄것을 부탁한다. 말단에서는 15명의 계산서기, 32명의 주문서기, 42명의 창고서기들이 작업한다. 18명의 관리자들이 자료에 접근할 필요가 있다. 사장은 하드웨어와 쏘프트웨어에 모두 3만딸라를 지불하려고 한다. 그리고 제품은 4주동안에 완성할것을 요구한다. 당신은 그에게 무엇이라고 말하겠는가? 고문으로서 그의 요구가 얼마나 리성이 없는가를 생각해 보시오.

1.3. 당신은 클라몬트공화국 해군중장이다. 다음세대 함선대함선미싸일을 위한 조종쏘프트웨어를 개발하기 위하여 어느 한 쏘프트웨어개발기업체를 요청하기로 결정하였다. 당신은 쏘프트웨어연구개발을 지휘할 책임을 지게 되였다. 클라몬트정부를 보위하기 위하여서는 쏘프트웨어개발자들과 계약하는데서 무슨 조항을 포함시켜야 하는가?

1.4. 당신은 프로그람기사이며 당신의 임무는 문제 1.3에서 제기된 쏘프트웨어개발을 지휘하는것이다. 당신의 회사가 해군과 한 계약을 만족시키는데서 실패할수 있는 정황들을 렬거하시오. 이러한 실패에 대하여 있을수 있는 원인을 분석하시오.

1.5. 배포후 15개월만에 내연기관에서 기름의 최량점성을 결정하는 기계공학쏘프트웨어제품에서 고장이 발견되였다. 고장을 퇴치하는데 18,730딸라가 들었다. 고장의 원인이 명세서에는 애매한 문구로 되여 있다. 명세서작성단계에서 오유를 퇴치하는데 대략 얼마만한 비용이 들겠는가?

1.6. 문제 1.5에서 제기된 고장이 실현단계에서 발견되였다고 하자. 그것을 퇴치하는데 대략 얼마만한 비용이 들겠는가?

1.7. 당신은 대규모쏘프트웨어를 개발하는 기관의 책임자이다. 당신은 그림 1-5를 종업원들에게 보여 주면서 쏘프트웨어생명주기에서 신속하게 오유를 찾아 내라고 재촉한다. 어떤 종업원이 제품개발에 들어 가기에 앞서 그 누군가가 오유를 제거하리라고 기대하는것은 타당치 않은 일이라고 대답했다. 실례로 질문에 제기된 오유가 코드작성에서의 오유이라면 설계단계에서 그 오유를 어떻게 퇴치할수 있는가? 당신은 무엇이라고 대답하겠는가?

1.8. 사전에서 단어 system을 찾아 보시오. 거기에 몇가지 의미가 있는가? 쏘프트웨어공학분야에 적용할수 있는 의미를 찾아 쓰시오.

1.9. 오늘은 당신이 일을 시작하는 첫날이다. 사장이 당신에게 목록을 보여 주며 《바그를 찾을수 있는가를 보시오.》라고 말했다. 무엇이라고 대답하겠는가?

1.10. 당신은 문제 1.1에서 제시한 제품을 개발할 책임을 지게 되였다. 객체지향파라다임을 리용하겠는가 또는 구조화파라다임을 리용하겠는가? 대답과 함께 그 리유를 밝히시오.

1.11. (과정안상 목표) 부록 1에 제시한 브로드랜즈지역 아동병원제품이 정확히 실현되였다고 하자. 이제 가족면회(JCF)프로그람은 브로드랜즈시의 926km 반경내에 사는 사람들만이 아니라 모든 환자들에게 적용된다고 하자. 현재 쏘프트웨어제품을 어떤 방법으로 변화시켜야 하는가? 처음부터 새로 시작하여도 되는가?

1.12. (쏘프트웨어공학독본) 교원은 [Reel, 1999]의 복사본을 배포할것이다. 제품오유에 대한 임의의 표식을 될수록 신속하게 발견할수 있도록 제품을 어떻게 관리할수 있는가?

참 고 문 헌

[Baetjer, 1996] H. BAETJER, *Software as Capital: An Economic Perspective on Software Engineering,* IEEE Computer Society Press, Los Alamitos, CA, 1996.

[Bhandari et al., 1994] I. BHANDARI, M. J. HALLIDAY, J. CHAAR, R. CHILLAREGE, K. JONES, J. S. ATKINSON, C. LEPORI-COSTELLO, P. Y. JASPER, E. D. TARVER, C. C. LEWIS, AND M. YONEZAWA, "In-Process Improvement through Defect Data Interpretation," *IBM Systems Journal* **33** (No. 1, 1994), pp. 182–214.

[Boehm, 1976] B. W. BOEHM, "Software Engineering," *IEEE Transactions on Computers* **C-25** (December 1976), pp. 1226–41.

[Boehm, 1979] B. W. BOEHM, "Software Engineering, R & D Trends and Defense Needs," in *Research Directions in Software Technology,* P. Wegner (Editor), The MIT Press, Cambridge, MA, 1979.

[Boehm, 1980] B. W. BOEHM, "Developing Small-Scale Application Software Products: Some Experimental Results," *Proceedings of the Eighth IFIP World Computer Congress,* October 1980, pp. 321–26.

[Boehm, 1981] B. W. BOEHM, *Software Engineering Economics,* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Brereton et al., 1999] P. BRERETON, D. BUDGEN, K. BENNETT, M. MUNRO, P. LAYZELL, L. MACAULAY, D. GRIFFITHS, AND C. STANNETT, "The Future of Software," *Communications of the ACM* **42** (December 1999), pp. 78–84.

[Brooks, 1975] F. P. BROOKS, JR., *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley, Reading, MA, 1975. Twentieth Anniversary Edition, Addison-Wesley, Reading, MA, 1995.

[Budd, 1991] T. A. BUDD, *An Introduction to Object-Oriented Programming,* Addison-Wesley, Reading, MA, 1991.

[Capper, Colgate, Hunter, and James, 1994] N. P. CAPPER, R. J. COLGATE, J. C. HUNTER, AND M. F. JAMES, "The Impact of Object-Oriented Technology on Software Quality: Three Case Histories," *IBM Systems Journal* **33** (No. 1, 1994), pp. 131–57.

[Coleman, Ash, Lowther, and Oman, 1994] D. COLEMAN, D. ASH, B. LOWTHER, AND P. OMAN, "Using Metrics to Evaluate Software System Maintainability," *IEEE Computer* **27** (August 1994), pp. 44–49.

[Daly, 1977] E. B. DALY, "Management of Software Development," *IEEE Transactions on Software Engineering* **SE-3** (May 1977), pp. 229–42.

[DeMarco and Lister, 1987] T. DEMARCO AND T. LISTER, *Peopleware: Productive Projects and Teams,* Dorset House, New York, 1987.

[DeMarco and Lister, 1989] T. DEMARCO AND T. LISTER, "Software Development: The State of the Art vs. State of the Practice," *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, May 1989, pp. 271–75.

[Ebert, Matsubara, Pezzé, and Bertelsen, 1997] C. EBERT, T. MATSUBARA, M. PEZZÉ, AND O. W. BERTELSEN, "The Road to Maturity: Navigating between Craft and Science," *IEEE Software* **14** (November/December 1997), pp. 77–88.

[El-Rewini et al., 1995] H. EL-REWINI, S. HAMILTON, Y.-P. SHAN, R. EARLE, S. MCGAUGHEY, A. HELAL, R. BADRACHALAM, A. CHIEN, A. GRIMSHAW, B. LEE, A. WADE, D. MORSE, A. ELMAGRAMID, E. PITOURA, R. BINDER, AND P. WEGNER, "Object Technology," *IEEE Computer* **28** (October 1995), pp. 58–72.

[Elshoff, 1976] J. L. ELSHOFF, "An Analysis of Some Commercial PL/I Programs," *IEEE Transactions on Software Engineering* **SE-2** (June 1976), 113–20.

[Fagan, 1974] M. E. FAGAN, "Design and Code Inspections and Process Control in the Development of Programs," Technical Report IBM-SSD TR 21.572, IBM Corporation, December 1974.

[Fichman and Kemerer, 1997] R. G. FICHMAN AND C. F. KEMERER, "Object Technology and Reuse: Lessons from Early Adopters," *IEEE Computer* **30** (July 1997), pp. 47–57.

[Finkelstein, 2000] A. FINKELSTEIN (EDITOR), *The Future of Software Engineering*, IEEE Computer Society Press, Los Alamitos, CA, 2000.

[Gibbs, 1994] W. W. GIBBS, "Software's Chronic Crisis," *Scientific American* **271** (September 1994), pp. 86–95.

[Glass, 1998] R. L. GLASS, "Is There Really a Software Crisis?" *IEEE Software* **15** (January/February 1998), pp. 104–5.

[Grady, 1994] R. B. GRADY, "Successfully Applying Software Metrics," *IEEE Computer* **27** (September 1994), pp. 18–25.

[IEEE 610.12, 1990] "A Glossary of Software Engineering Terminology," IEEE 610.12-1990, Institute of Electrical and Electronic Engineers, Inc., New York, 1990.

[ISO/IEC 12207, 1995] "ISO/IEC 12207:1995, Information Technology—Software Life-Cycle Processes," International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Johnson, 2000] R. A. JOHNSON, "The Ups and Downs of Object-Oriented System Development," *Communications of the ACM* **43** (October 2000), pp. 69–73.

[Josephson, 1992] M. JOSEPHSON, *Edison: A Biography*, John Wiley and Sons, New York, 1992.

[Kan et al., 1994] S. H. KAN, S. D. DULL, D. N. AMUNDSON, R. J. LINDNER, AND R. J. HEDGER, "AS/400 Software Quality Management," *IBM Systems Journal* **33** (No. 1, 1994), pp. 62–88.

[Kelly, Sherif, and Hops, 1992] J. C. KELLY, J. S. SHERIF, AND J. HOPS, "An Analysis of Defect Densities Found during Software Inspections," *Journal of Systems and Software* **17** (January 1992), pp. 111–17.

[Khan, Al-A'ali, and Girgis, 1995] E. H. KHAN, M. AL-A'ALI, AND M. R. GIRGIS, "Object-Oriented Programming for Structured Procedural Programming," *IEEE Computer* **28** (October 1995), pp. 48–57.

[Kroeker et al., 1999] K. K. KROEKER, L. WALL, D. A. TAYLOR, C. HORN, P. BASSETT, J. K. OUSTERHOUT, M. L. GRISS, R. M. SOLEY, J. WALDO, AND C. SIMONYI, "Software [R]evolution: A Roundtable," *IEEE Computer* **32** (May 1999), pp. 48–57.

[Leveson, 1997] N. G. LEVESON, "Software Engineering: Stretching the Limits of Complexity," *Communications of the ACM* **40** (February 1997), pp. 129–31.

[Leveson and Turner, 1993] N. G. LEVESON AND C. S. TURNER, "An Investigation of the Therac-25 Accidents," *IEEE Computer* **26** (July 1993), pp. 18–41.

[Lewis, 1996a] T. LEWIS, "The Next $10{,}000_2$ Years: Part I," *IEEE Computer* **29** (April 1996), pp. 64–70.

[Lewis, 1996b] T. LEWIS, "The Next $10{,}000_2$ Years: Part II," *IEEE Computer* **29** (May 1996), pp. 78–86.

[Lientz, Swanson, and Tompkins, 1978] B. P. LIENTZ, E. B. SWANSON, AND G. E. TOMPKINS, "Characteristics of Application Software Maintenance," *Communications of the ACM* **21** (June 1978), pp. 466–71.

[Littlewood and Strigini, 1992] B. LITTLEWOOD AND L. STRIGINI, "The Risks of Software," *Scientific American* **267** (November 1992), pp. 62–75.

[Maring, 1996] B. MARING, "Object-Oriented Development of Large Applications," *IEEE Software* **13** (May 1996), pp. 33–40.

[Mellor, 1994] P. MELLOR, "CAD: Computer-Aided Disaster," Technical Report, Centre for Software Reliability, City University, London, U.K., July 1994.

[Meyer, 1992a] B. MEYER, "Applying 'Design by Contract'," *IEEE Computer* **25** (October 1992), pp. 40–51.

[Meyer, 1997] B. MEYER, *Object-Oriented Software Construction,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1997.

[Naur, Randell, and Buxton, 1976] P. NAUR, B. RANDELL, AND J. N. BUXTON (Editors), *Software Engineering: Concepts and Techniques: Proceedings of the NATO Conferences,* Petrocelli-Charter, New York, 1976.

[Neumann, 1980] P. G. NEUMANN, Letter from the Editor, *ACM SIGSOFT Software Engineering Notes* **5** (July 1980), p. 2.

[Neumann, 1995] P. G. NEUMANN, *Computer-Related Risks,* Addison-Wesley, Reading, MA, 1995.

[Parnas, 1990] D. L. PARNAS, "Education for Computing Professionals," *IEEE Computer* **23** (January 1990), pp. 17–22.

[Parnas, 1994] D. L. PARNAS, "Software Aging," *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 279–87.

[Radin, 1996] G. RADIN, "Object Technology in Perspective," *IBM Systems Journal* **35** (No. 2, 1996), pp. 124–126.

[Reel, 1999] J. S. REEL, "Critical Success Factors in Software Projects," *IEEE Software* **16** (May/June 1999), pp. 18–23.

[Scholtz et al., 1993] J. SCHOLTZ, S. CHIDAMBER, R. GLASS, A. GOERNER, M. B. ROSSON., M. STARK, AND I. VESSEY, "Object-Oriented Programming: The Promise and the Reality," *Journal of Systems and Software* **23** (November 1993), pp. 199–204.

[Shapiro, 1994] F. R. SHAPIRO, "The First Bug," *Byte* **19** (April 1994), p. 308.

[Shneiderman, 1980] B. SHNEIDERMAN, *Software Psychology: Human Factors in Computer and Information Systems,* Winthrop Publishers, Cambridge, MA, 1980.

[Stephenson, 1976] W. E. STEPHENSON, "An Analysis of the Resources Used in Safeguard System Software Development," Bell Laboratories, Draft Paper, August 1976.

[Wasserman, 1996] A. I. WASSERMAN, "Toward a Discipline of Software Engineering," *IEEE Software* **13** (November/December 1996), pp. 23–31.

[Webster, 1995] B. F. WEBSTER, *Pitfalls of Object-Oriented Development,* M&T Books, New York, 1995.

[Weinberg, 1971] G. M. WEINBERG, *The Psychology of Computer Programming,* Van Nostrand Reinhold, New York, 1971.

[Wirfs-Brock, Wilkerson, and Wiener, 1990] R. WIRFS-BROCK, B. WILKERSON, AND L. WIENER, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[Yourdon, 1996] E. YOURDON, *Rise and Resurrection of the American Programmer,* Yourdon Press, Upper Saddle River, NJ, 1996.

[Zelkowitz, Shaw, and Gannon, 1979] M. V. ZELKOWITZ, A. C. SHAW, AND J. D. GANNON, *Principles of Software Engineering and Design,* Prentice Hall, Englewood Cliffs, NJ, 1979.

[Zvegintzov, 1998] N. ZVEGINTZOV, "Frequently Begged Questions and How to Answer Them," *IEEE Software* **15** (January/February 1998), pp. 93–96.

제2장. 쏘프트웨어개발공정

쏘프트웨어개발공정은 쏘프트웨어를 생산하는 과정이다. 그것은 쏘프트웨어생명주기모형(1.3)과 우리가 리용하게 되는 도구들(5.4~5.10), 쏘프트웨어를 구성하는데서 중요한 개별적인 요소들을 모두 포함하고 있다.

매개 개발기업체들마다 쏘프트웨어개발공정이 서로 다르다. 실례로 문서작성문제를 생각해 보자. 일부 기업체들은 자체로 문서를 작성할수 있도록 하는 쏘프트웨어를 연구한다. 즉 그러한 제품은 원천코드를 읽어 보고 간단히 리해할수 있다. 그러나 다른 기업체들은 문서화에 집중한다. 그들은 깐깐하게 명세를 작성하고 그 명세를 방법론적으로 검사한다. 그다음 주의 깊게 설계를 진행하며 코드작업을 진행하기전에 설계를 검사하고 또 검사한다. 그다음 프로그람작성자들에게 매 모듈에 대하여 상세한 설명을 해준다. 시험실례들은 미리 계획되며 매 시험결과들이 기록되며 시험자료를 주의 깊게 파일로 작성한다. 일단 제품이 유지정비단계에 들어 가면 모든 임의의 제기된 변경들은 변경을 진행하여야 할 상세한 리유와 함께 서면으로 제출되여야 한다. 제출된 변경은 오직 주어 진 권한내에서만 진행할수 있으며 문서가 갱신되고 문서에 대한 변경이 승인될 때까지는 변경이 제품에 통합되지 않는다.

시험의 강도는 개발기업체들이 비교될수 있는 또 다른 척도로 된다. 일부 기업체들이 쏘프트웨어시험에 쏘프트웨어개발예산의 절반까지 소비하는 반면에 다른 기업체들은 사용자들만이 제품을 철저히 시험할수 있다고 생각하고 있다. 결국 일부 회사들은 제품을 시험하는데 최소한의 시간과 품을 들이고 있으나 사용자들이 제기하는 문제들을 수정하는데는 상당한 시간을 들이고 있다. 유지정비는 많은 쏘프트웨어기업체들에서 첫째로 중요한 임무로 되고 있다. 5년, 10년 지어는 20년전에 만든 낡은 쏘프트웨어는 요구가 변화되는데 따라 계속 강화되게 된다. 그리고 쏘프트웨어가 여러해동안 성공적으로 유지정비되여 온 다음에도 잔류오유들이 계속 나타난다. 거의 모든 기업체들은 3~5년마다 한번씩 새로운 하드웨어에 쏘프트웨어를 넘겨 준다. 이것 역시 유지정비를 구성하게 된다.

이와 대조적으로 아직도 일부 기업체들은 개발은 다른 기업체들에 맡기고 유지정비만을 허용하는 연구에 집중하고 있다. 이 방법은 특히 대학의 콤퓨터과학학부들에서 응용되고 있는데 여기서는 연구생들이 특정한 설계나 기법이 실현가능하다고 증명하기 위한 쏘프트웨어를 구성한다. 공인된 개념들에 대한 상업적인 리용은 다른 기업체들에 맡겨 진다(다른 기업체들에서 쏘프트웨어를 개발하고 있는 방법들에 대해서는 다음의 《알고 싶은 문제》를 보시오.).

그러나 정확한 절차를 무시한다고 하여도 쏘프트웨어개발공정은 크게 1.3에서 서술한 7개의 단계 즉 요구사항확정, 명세작성, 설계, 실현, 통합, 유지정비와 페기를 거치게 된다. 이 단계들의 일부는 다른 이름으로 알려 져 있다. 실례로 요구사항확정, 명세작성을 합쳐서 체계분석(*systems analysis*)이라고 부르기도 한다.

알고 싶은 문제

왜 쏘프트웨어개발공정은 기업체마다 완전히 다른가? 그 하나의 리유는 프로그람공학에서의 기능이 부족하기때문이다. 많은 쏘프트웨어전문가들이 최신기술을 소유하지 못하고 있다. 그들은 그 어떤 다른 방도는 모르기때문에 여전히 Ye Olde Fashioned Way식으로 쏘프트웨어를 개발하고 있다.

쏘프트웨어개발공정에서 차이가 있게 되는 또 다른 리유는 많은 쏘프트웨어관리자들이 우수한 관리자로는 되지만 쏘프트웨어개발이나 유지정비에 대하여서는 거의나 알고 있지 못하고 있기때문이다. 그들이 이처럼 기술지식이 결여되여 있는것으로 하여 프로젝트가 일정계획에서 심히 리탈하여 더는 프로젝트수행을 계속해 나갈수 없게 된다. 이것이 흔히 많은 쏘프트웨어프로젝트들이 완성되지 못하는 리유로 되고 있다.

개발공정들사이에서 차이나는 또 다른 리유는 관리에 대한 견해에서의 차이에 있기때문이다. 실례로 어떤 기업체는 제품이 비록 충분히 시험되지 않았다고 하여도 제때에 배포하는것이 더 좋다고 결정할수 있다. 같은 정황하에서 다른 기업체들은 종합적인 시험을 진행하지 않고 그 제품을 배포하는것이 제품을 철저히 시험한 다음에 배포하는것보다 훨씬 더 큰 위험요소들을 포함하고 있다고 결론할수도 있다. 요점은 쏘프트웨어가 인간에 의하여 개발되고 해당한 기업체에서 진행하는 개발공정이 그 기업체에서 일하는 개별적인 사람들과 관련된다는것이다. 만일 그러한 개별적인 사람들이 의리가 있고 근면하고 지적이며 첨단기술을 소유하고 있다면 그 기업체에서 진행하는 쏘프트웨어개발공정은 만족스럽게 진행될것이다. 유감스럽게도 그 반대의 경우도 있을수 있다.

더우기 어떤 단계들은 부분적으로 나누어 진행될수 있다. 실례로 설계단계는 거의 언제나 구성방식설계(*architectural design*)와 상세설계(*detailed design*)로 분할된다. 앞에서 렬거한 단계들에서 시험단계는 따로 떨어 져 있지 않다. 이것이 빠져 있는것은 심중하게 고려한것이다. 시험은 따로 떨어 진 단계가 아니라 쏘프트웨어생산의 모든 공정들에서 진행되는 사업이다. 요구사항도 시험되여야 하고 명세서도 시험되여야 하며 설계도 시험되여야 하는 등 시험이 진행되여야 한다. 개발공정에서는 다른 사업을 거의나 모두 제외하고 시험을 진행하여야 할 때가 있다. 이것은 매 단계의 마감(검증)에서 발생하며 특히 제품이 의뢰자들에게 넘어 가기전에 진행되여야 한다(확인). 비록 시험이 우세를 차지할 때는 있어도 시험이 진행되지 않는 때란 절대로 있을수 없다. 만일 시험이 따로 떨어 져 있는 단계로 취급된다면 제품개발과 유지정비과정을 비롯한 매 단계들에서 항상 시험이 진행되지 않는것으로 하여 실제적인 위험이 조성되게 된다. 또한 문서작성도 따로 떨어 진 단계가 아니다. 그것은 바로 다음단계를 시작하기전에 매 단계를 완전히 문서화해야 하기때문이다.

1. 어떤 제품을 제때에 배포하여야 한다는것은 바로 문서작성을 뒤로 미루고서는 제품을 절대로 완성할수 없다는것이다.
2. 쏘프트웨어개발공정의 어떤 초기단계에 책임이 있는 개별적사람은 문서가 작성되여야 할 때 다른 부문을 책임지거나 다른 기업체들을 위하여 사업할수도 있다.
3. 제품은 개발되는 기간 끊임없이 변화된다. 실례로 설계는 보통 제품에 대한 새

로운 정보를 고려하여 실현단계에서 수정되여야 한다. 설계가 설계팀에 의하여 완전히 문서화되지 못하면 그것에 대한 수정은 아주 어렵게 된다.

4. 원래의 설계자들은 설계가 수정된 다음 그것을 문서화하기 어렵게 된다.

이런 리유로 하여 쏘프트웨어개발공정의 매 단계를 위한 문서는 다음단계를 시작하기전에 그 단계에 책임이 있는 팀이 완성하여야 한다. 더우기 문서는 제품의 현재 판본을 반영하도록 계속 갱신하여야 한다. 이 장에서는 제품이 걸쳐야 할 단계들을 서술하고 그와 함께 매 단계에서 제기될수 있는 가능한 난점들에 대하여 서술한다. 이밖에 매 단계에 적합한 시험절차도 서술한다. 쏘프트웨어생산과 관련된 난점들을 해결하는것은 일반적으로 중요하다. 그리고 이 책의 나머지 부분들에서는 적당한 기법들에 대하여 서술하고 있다. 이 장의 첫부분에서 다만 난점들을 강조하였지만 독자들은 그 해결과 관련된 절들이나 장들을 안내 받게 된다. 따라서 이 부분에서는 쏘프트웨어개발공정에 대한 개괄뿐아니라 이 책의 나머지 부분들에 대한 안내를 많이 주고 있다.

매개 단계들과 관련된 문제들에 대하여 이러한 설명을 한 다음 쏘프트웨어생산에서 제기되는 고유한 난점들을 전반적으로 서술하였다. 이 장은 쏘프트웨어개발공정을 개선하기 위한 국내 및 국제적발전추세에 대한 론의로 결속된다.

2. 1. 의뢰자, 개발자, 사용자

여기서는 몇가지 예비적인 정의가 필요하다. 의뢰자(*client*)는 어떤 제품을 개발해 줄것을 요구하는 개별적사람 또는 기업체이다. 개발자(*developer*)는 그러한 제품개발에 책임이 있는 기업체의 성원이다. 개발자는 요구사항확정단계로부터 시작하여 개발공정의 모든 측면에 대하여 책임을 지거나 또는 임의의 설계된 제품의 실현에 대하여서만 책임을 질수 있다. 용어 쏘프트웨어개발(*software development*)은 제품이 유지정비단계에 들어 가기전 쏘프트웨어생산의 모든 측면을 포함한다. 명세작성과 계획작성, 설계, 시험 또는 문서작성을 비롯한 쏘프트웨어의 부분품을 개발해 나가는 매 단계의 과제들은 쏘프트웨어개발들로 구성되여 있다. 쏘프트웨어는 개발된 다음 유지정비되여야 한다.

의뢰자와 개발자는 다 같은 기업체에 속할수 있다. 실례로 의뢰자는 보험회사의 서기일수도 있고 개발자는 그 보험회사의 관리정보체계를 맡아 보는 부사장소속의 한 성원일수도 있다. 이것을 내부쏘프트웨어(*internal software*)개발이라고 부른다. 다른 한편 계약식쏘프트웨어(*contract software*)가 있는데 이때 의뢰자와 개발자는 완전히 독립적인 기업체들로 된다. 실례로 의뢰자는 방위성의 장관일수도 있고 개발자들은 무기체계용쏘프트웨어를 전문으로 하는 주요방위쏘프트웨어계약자들일수 있다. 훨씬 더 작은 규모에서 말하면 의뢰자는 한사람이 진행하는 업무에서의 부기원일수 있으며 개발자는 시간제로임으로 쏘프트웨어를 개발해 주고 수입을 얻는 대학생일수도 있다.

쏘프트웨어생산에 참가하는 세번째 대상은 사용자이다. 사용자는 의뢰자들이 제품을 주문하는데 리해관계를 가지면서 그 쏘프트웨어를 리용할 사람들이다. 보험회사의 실례에서 사용자는 쏘프트웨어를 리용하여 가장 적합한 방안을 선택해 나가는 보험대리인들이다. 일부 실례들에서는 의뢰자이자 사용자로 될수 있다(실례로 앞에서 이야기한 부기

원을 들수 있다.).

한명의 의뢰자를 위하여 만들어 진 값 비싼 쏘프트웨어와는 반대로 문서처리기나 표처리프로그람과 같은 쏘프트웨어들은 여러개 복사되여 수많은 구매자들에게 눅은 가격으로 팔리게 된다. 이러한 쏘프트웨어제작자(마이크로쏘프트회사나 볼랜드회사와 같은)들은 대량판매로 제품을 개발하는데 드는 가격을 보상하고 있다. 이러한 류형의 쏘프트웨어는 보통 상업용기성(*commerical off-the-shelf*; COTS)쏘프트웨어라고 부른다. 이러한 쏘프트웨어에 대한 초기의 학술용어는 수축포장쏘프트웨어(*shrink-wrapped software*)였는데 그것은 CD나 디스케트, 설명서 그리고 사용허가신청서 등을 포함한 통들이 거의나 수축포장되기때문이다. 요즘 COTS쏘프트웨어는 흔히 WWW에서 내리적재되고 있으며 수출포장통은 없어 졌다. 이런 리유로 하여 COTS쏘프트웨어는 최근에 때때로 클리크포장쏘프트웨어(*click-wrapped software*)라고 부르고 있다. COTS쏘프트웨어는 《시장》을 대상하여 개발된다. 즉 쏘프트웨어가 개발되여 사서 쓸수 있게 될 때까지는 특정한 의뢰자나 사용자가 없게 된다. 우리는 이 장의 다음부분에서 쏘프트웨어생명주기의 7개 단계에 대하여 서술하며 매 단계를 시험하여 그 역할을 주의 깊게 분석한다. 첫 단계는 요구사항확정단계이다.

2. 2. 요구사항확정단계

쏘프트웨어개발은 품이 드는 공정이다. 개발공정은 보통 의뢰자의 견해에 따라 기업의 리득이라든가 또는 그 어떤것이 경제적으로 정당화될수 있는 쏘프트웨어제품에 주목하여 의뢰자들이 개발기업체들에 의뢰하게 될 때 시작되게 된다. 개발공정의 임의의 단계에서 의뢰자들이 쏘프트웨어가 비용상 효률적이라는것을 믿지 못할 때는 개발은 즉시에 정지되고 만다. 이 장에서는 의뢰자들이 비용이 타당하다고 간주한다는것을 념두에 둔다(사실상 《비용》은 언제나 순수한 재정상의 문제로 되는것이 아니다. 실례로 군사분야의 쏘프트웨어는 흔히 전략적인 또는 전술적인 목적으로 만들어 진다. 여기서 쏘프트웨어의 비용은 바로 개발되고 있는 무기가 없는것으로 하여 받게 되는 잠재적인 위험으로 된다.). 의뢰자가 개발자와 처음 만나면 의뢰자는 제품을 개념화하여 륜곽적으로 설명한다. 개발자의 견지에서는 요구하는 제품에 대한 의뢰자의 설명이 명백치 않거나 불합리하며 모순적이고 단순히 실현불가능할수도 있다. 이 단계에서 개발자들의 과제는 의뢰자가 무엇을 요구하고 어떤 제약이 있는가를 의뢰자로부터 정확히 밝혀 내는것이다. 전형적인 제약은 제품을 완료하는 마감기한이다. 실례로 의뢰자는 완전한 제품이 14개월안으로 완성되여야 한다고 규정할수 있다. 기타 여러가지 제약들은 흔히 신뢰성(제품은 99%에 해당한 시간동안 가동하여야 한다.)이나 객체의 크기(그것은 의뢰자의 콤퓨터상에서 동작하여야 한다.)로 나타낼수 있다.

일반적으로 가장 중요한 제한은 비용이다. 그러나 의뢰자는 제품을 개발하는데 얼마만한 비용이 드는가를 개발자에게 좀처럼 말해 주지 않는다. 그대신에 일반적으로 일단 명세작성이 끝나면 의뢰자는 개발자들에게 프로젝트를 완성하기 위한 가격을 물어 본다. 의뢰자는 개발자들이 부른 비용이 자기들이 그 프로젝트에 대하여 세운 예산량보다 더

적을것이라는 기대를 가지고 이러한 값부르기절차에로 넘어 간다. 의뢰자들의 요구에 따라 진행하는 이런 초보적인 조사를 흔히 개념탐구(*concept exploration*)라고 한다. 개발팀과 의뢰자팀성원들이 계속 만나는 과정에 제기된 제품의 기능이 성과적으로 완성되고 기술적인 실현가능성과 재정상의 타당성이 분석된다.

지금까지는 모든것이 단순한것 같다. 유감스럽게도 요구사항확정단계는 흔히 부적당하게 수행된다. 제품이 최종적으로 사용자들에게 배포되면 아마 의뢰자는 명세서에 수표한 다음 1년 또는 2년후에 개발자들에게 《내 알기에는 이것이 내가 요청한것이지만 사실은 그것이 내가 바라던것은 아니다.》라고 말할수 있다. 결국 의뢰자들이 요청하였고 따라서 개발자들이 의뢰자가 바라던것이라고 생각한것은 의뢰자들이 사실상 요구한것이 아니였다. 이런 상태에 처하게 된데는 여러가지 리유가 있을수 있다. 우선 의뢰자들은 자기의 기업체들에서 진행하고 있는것을 사실상 리해하지 못하고 있을수 있다. 실례로 현재 일감처리가 느리게 진행되는 원인이 자료기지가 잘못 설계되였기때문이라면 쏘프트웨어개발자들에게 보다 빠른 조작체계를 요청할 필요는 없다. 또는 의뢰자가 수지가 맞지 않는 련쇄소매창고를 관리한다고 하면 판매와 로임, 지불계산, 수입계산 등과 같은 항목들을 반영하는 재정경영정보체계를 요청할수 있다. 만일 손실을 보는 실제적인 원인이 감소(상점에서의 좀도적과 종업원들에 의한 절취)에 있다면 그런 제품은 리용가치가 거의나 없다. 이런 경우라면 재정관리제품이 아니라 창고관리제품이 요구된다.

그러나 의뢰자들이 흔히 제품을 잘못 요청하게 되는 기본원인은 쏘프트웨어가 복잡하기때문이다. 만일 쏘프트웨어전문가가 쏘프트웨어의 일부분과 그 기능을 시각화하기가 곤난하다면 거의나 콤퓨터에 대하여 파악이 없는 의뢰자에게 있어서 문제는 더 악화될것이다. 이에 대처하기 위한 여러가지 방법들이 있는데 그중의 하나가 신속원형작성이다.

신속원형(*rapid prototype*)은 서둘러 모아 놓은 프로그람의 부분인데 그것은 목표제품의 대부분의 기능들은 병합하고 있지만 파일갱신이나 오유조종과 같은 의뢰자들에게 보통 보이지 않는 측면들은 빠뜨려 놓고 있다. 그러면 의뢰자와 사용자는 그것이 자기들의 요구에 부합되는가를 결정하기 위하여 원형을 가지고 실험을 진행한다. 신속원형은 의뢰자와 사용자들이 자기들의 요구하는 기능을 교갑화한다는것을 확인할 때까지 변화될수 있다. 신속원형작성과 기타 다른 요구사항분석기법은 제10장에서 상세히 론의된다.

2. 2. 1. 요구사항확정단계에서의 시험

매 쏘프트웨어개발기업체들에는 배포된 제품이 의뢰자들이 주문한것이고 제품이 모든 면에서 정확이 개발되였다는것을 담보하는것을 책임지는 그룹이 있어야 한다. 이러한 그룹을 쏘프트웨어품질보증(*software quality assurance*; SQA)그룹이라고 부른다. 쏘프트웨어의 품질은 쏘프트웨어의 명세서에 반영된 범위내에서 담보된다. 품질과 쏘프트웨어품질보증에 대해서는 6장에서 자세히 서술되는데 SQA는 표준을 설정하고 집행하는 역할을 수행한다.

SQA그룹은 개발공정의 시작부터 역할을 옳바로 수행하여야 한다. 특히 제품이 의뢰자들의 요구를 만족시키는것은 사활적인 문제이다. 따라서 SQA그룹은 신속원형에 대한 마지막 판본이 전적으로 만족한다는것을 의뢰자와 함께 확증하여야 한다.

제품이 자기들의 요구를 반영하고 있다는것을 확증하도록 의뢰자와 사용자가 다같이 주의 깊게 신속원형을 검사하는것이 중요하다. 아무리 세심하게 검사가 진행된다 하더라도 제품이 개발되고 있는 동안 개발팀을 조종하는 세력이 요구사항의 변화를 요구할 가능성이 항상 존재한다. 더우기 부분적으로 완성된 제품에 대하여 필요한 변경이 진행될때까지는 개발이 유지되여야 한다.

쏘프트웨어개발에서 기본문제는 이른바 목표가 이동하는 문제이다. 즉 개발기간 의뢰자의 요구사항이 변화되는것이다. 이렇게 되는 리유의 하나는 주위환경에서 뜻하지 않은 우연적인 변화가 일어 난다는데 있다. 실례로 어떤 회사가 자기 사업을 확장하거나 또는 다른 회사에 채용된다면 그때는 여전히 개발상태에 있는 제품들을 포함하여 많은 제품들을 변경하여야 한다. 그러나 이동하는 목표문제의 기본원인은 자기들의 요구를 바꾸는 의뢰자들에게 있다. 16.4.4에서 설명한바와 같이 의뢰자가 충분한 권한을 가지고 있다면 어찌할 도리가 없다.

2. 2. 2. 요구사항확정단계에서의 문서작성

요구사항확정단계에서 만들어 진 문서는 일반적으로 신속원형과 구성변경된 신속원형이 기초하고 있는 의뢰자와 사용자들의 론의에 대한 기록을 포함하고 있다. 만일 팀이 신속원형을 만들지 않기로 결심하였다면 의뢰자의 요구를 서술하는 요구문서가 작성되게 된다. 이 문서는 의뢰자와 선택된 사용자들, 개발팀에 의하여 SQA그룹이 그것을 면밀히 조사하기전에 검열되여야 한다.

2. 3. 명세작성단계

일단 의뢰자가 개발자들이 요구사항을 리해하였다는것을 인정하면 명세서(*specification document*)는 명세작성팀에 의하여 작성된다. 비형식적요구사항확정단계와는 반대로 명세서는 그 제품의 기능 즉 정확히는 그 제품이 무엇을 하기로 되여 있는가를 명백히 서술하고 있으며 그 제품이 만족해야 할 제약들을 렬거하고 있다. 명세서는 제품에 대한 입력과 요구되는 출력을 포함한다. 실례로 의뢰자가 로임지불제품을 필요로 한다면 그때 입력에는 세금이 정확히 계산될수 있도록 개인자료철로부터 얻을수 있는 정보는 물론 매 종업원들의 지불량과 로동시간에 대한 자료를 포함된다. 출력은 로임지불행표와 보고서이다. 이밖에 명세서는 의료보험비나 동맹비 그리고 년료보장비와 같은 넓은 범위의 공제액을 정확히 관리할수 있는 조항들을 포함한다.

제품의 명세서는 하나의 제약을 구성한다. 쏘프트웨어개발자들은 명세서의 접수기준을 만족시키는 제품을 배포할 때 계약을 완수하였다고 생각한다. 이런 리유로 하여 명세서에는 적합한(*suitable*), 편리한(*convenient*), 풍부한(*ample*) 또는 충분한(*enough*)과 같은 정확치 못한 용어들이나 최량적인(*optimal*) 또는 98% 완전한(98% *percent complete*)과 같이 정확하지만 실천적으로는 부정확한 용어들을 포함하지 말아야 한다. 쏘프트웨어개발계약은 소송에 제기될수 있는 반면에 의뢰자와 개발자들이 같은 기업체에 있을 때에는 명세

서가 법률상의 사업의 기초로 될 경우는 없게 된다. 그러나 내부쏘프트웨어개발의 경우에조차 명세서는 언제나 그것이 재판의 증거로 리용되는것처럼 작성되여야 한다. 보다 중요한것은 명세서가 시험과 유지정비에서 모두 중요하다는것이다.

명세서가 정확하지 않는 한 명세의 정확성여부를 결정할수 없으며 하물며 실현이 명세서를 만족시키는가 못시키는가를 결정할수 없다. 만일 명세들을 정확히 서술한 문서가 없으면 유지정비단계에서 명세서를 변경시키는것이 어렵게 된다.

명세작성단계에서 여러가지 난점들이 제기될수 있다. 명세작성팀에 의하여 제기될수 있는 오유의 하나는 명세서가 애매하다는것 즉 어떤 문장이나 부분들이 하나이상의 타당한 해석을 가지고 있다는것이다. 다음의 명세서를 고찰해 보자.

《하나의 부분기록과 하나의 식물기록이 자료기지로부터 읽어 진다. 만일 그것이 문자 A에 직접 뒤따르는 문자 Q를 포함하고 있다면 그때에 그 부분기록을 그 식물기록에로 옮기는데 드는 비용을 계산하시오.》

우의 문장에서 그것이 무엇으로 간주되는가 즉 부분기록인가 아니면 식물기록인가? 사실상 그것은 아마도 자료기지를 의미할수 있다.

명세서는 또한 불완전할수도 있다. 즉 일부 련관된 사실이나 요구사항들을 빠뜨릴수도 있다. 실례로 명세서는 입력자료가 오유를 포함하게 되면 무슨 작용이 일어 나겠는가를 서술하지 않을수도 있다. 더우기 명세서에는 모순도 있을수 있다. 실례로 발효과정을 조종하는 제품에 대한 명세서의 한 곳에는 압력이 35psi을 넘으면 발브 M17이 닫겨 져야 한다는것을 서술하고 있다. 또 다른 곳에는 압력이 35psi을 넘으면 그때에는 조작기에 즉시 경고신호를 보내야 한다고 서술하고 있다. 즉 조작기가 30s이내에 보수되지 않으면 발브 M17은 자동적으로 닫겨 져야 한다는 경고가 서술되고 있다. 명세서에서의 이러한 문제들이 정정되지 않는한 쏘프트웨어개발공정은 진행될수 없다.

일단 명세서가 완성되면 세부계획작성과 타산이 시작된다. 프로젝트를 개발완성하는데 기간이 얼마나 걸리고 비용이 얼마나 드는가 하는것을 미리 알지 못하고서는 그 어떤 의뢰자도 쏘프트웨어프로젝트를 허가하지 않을것이다. 개발자들의 견지에서 이 두 항목은 아주 중요하다. 만일 개발자들이 프로젝트의 가격을 낮게 정한다면 의뢰자들은 합의한 가격으로 지불하게 되는데 그 가격은 개발자들에게 있어서 실제가격보다 훨씬 더 적은 값일수 있다. 반대로 개발자들이 프로젝트의 가격을 높이 정한다면 의뢰자들은 그 프로젝트를 포기하거나 또는 값을 보다 적당하게 결정하는 다른 개발자들에게 그 일감을 맡기게 된다. 만일 개발자들이 프로젝트를 완성하는데 걸리는 시간을 과소평가한다면 제품을 늦게 배포하는것으로 하여 의뢰자들의 신용을 잃게 된다. 최악의 경우에는 계약에 따라 지연에 대한 벌금이 부과되여 개발자들은 재정상 곤경을 겪게 된다. 또한 개발자들이 배포될 제품을 개발하는데 걸리는 시간을 과대평가한다면 의뢰자들은 보다 빨리 제품을 배달할것을 약속하는 다른 개발자들에게 일감을 맡기게 될것이다. 개발자들이 개발기간과 전체 가격을 타산하는것은 충분하지 않다. 개발자들은 적당한 사람들에게 개발공정의 여러단계를 맡겨 주어야 한다. 실례로 코드작성팀은 설계문서가 SQA그룹에 의하여 인정되기전에는 코드작성을 시작할수 없으며 명세작성팀이 자기 과제를 완성하기전에는 설계팀이 필요 없게 된다. 달리 말하면 개발자들은 앞서나가면서 계획을 작성하게 된다. 쏘프트웨어프로젝트관리계획(SPMP)은 개발공정의 개별적인 단계를 반영하며 매 과제를

완성하기 위한 최종기간은 물론 매 개발기업체안의 누가 매 과제에 포함되는가를 보여주도록 작성되여야 한다.

이런 구체적인 계획이 작성될수 있는 가장 이른 시기는 명세서가 완성되는 때이다. 그전에는 프로젝트가 너무나도 막연하여 완성된 계획작성에 착수할수 없다. 프로젝트의 어떤 측면은 반드시 시작부터 잘 계획되여야 하지만 개발자들이 무엇이 개발되는가를 정확히 알 때까지는 그것을 개발하기 위한 계획의 모든 측면을 명시할수는 없다.

따라서 일단 명세서가 완성되여 검열되면 쏘프트웨어프로젝트관리계획의 준비가 시작된다. 계획의 기본구성부분은 배포물(의뢰자들이 얻으려 하는것), 리정표(의뢰자들이 언제 얻는가)와 예산안(비용이 얼마나 드는가)으로 된다.

계획은 아주 상세하게 쏘프트웨어개발공정을 서술한다. 그것은 리용되여야 할 생명주기모형이나 개발기업체의 조직구성, 프로젝트의 책임성, 경영목적과 우선권, 리용되게 될 기법과 CASE도구, 세부일정계획, 예산안, 자원할당과 같은 측면들을 포함한다. 전반적인 계획의 기초로 되는것은 개발기간과 비용에 대한 타산이다. 이러한 타산을 내릴수 있는 기법은 9.2에서 서술하고 있다. 명세작성단계는 11장과 12장에서 서술하고 있다. 즉 고전적인 기법은 11장에서, 객체지향분석은 12장의 주제이다(용어 **분석**(*analysis*)은 때때로 명세작성단계에서의 행동을 나타내는데 리용되며 따라서 **객체지향분석**(*object oriented analysis*)단계에서 리용된다.).

2. 3. 1. 명세작성단계에서의 시험

1장에서 언급한바와 같이 배포된 쏘프트웨어에서 생기는 오유의 기본원천은 쏘프트웨어가 의뢰자의 콤퓨터에 설치되여 의도하는 목적으로 의뢰자들의 기업체에서 리용될 때까지 발견되지 않는 명세서에서의 오유들이다. 따라서 SQA그룹은 명세서를 주의 깊게 살피면서 모순이 있는것, 애매한것 그리고 불완전한 기호들을 찾아 보아야 한다. 이밖에 SQA그룹은 명세서가 실현가능하다는것을 담보해야 한다. 실례로 어떤 특정한 하드웨어 구성요소가 충분히 빠르다는것 또한 의뢰자의 현재 직결디스크측정용량이 새로운 제품을 관리하는데 충분하다는것 등을 담보해야 한다. 만일 어떤 명세서가 시험가능하다면 그것이 가져야 할 속성의 하나는 추적가능성(*traceability*)이다. 추적가능성이란 요구사항확정단계에서 의뢰자팀이 작성한 설명문들로부터 명세서에 있는 매 설명문들을 추적할수 있어야 한다는것이다. 만일 요구사항이 방법론적으로 서술되고 정확히 번호화되여 있고 교차참조하도록 되여 있으며 첨수화되였다면 SQA그룹이 명세서를 추적하여 그것이 실지로 의뢰자의 요구를 반영하고 있다는것을 담보하는것이 거의나 어렵지 않다. 만일 신속원형작성이 요구사항확정단계에서 리용되면 련관된 명세서의 설명문들은 신속원형에 대하여 추적할수 있어야 한다.

명세서를 검열하는 가장 좋은 방법은 심사이다. 명세작성팀과 의뢰자팀의 대표자들이 참가한다. 면담은 보통 SQA그룹성원들에 의하여 집행된다. 심사의 목적은 명세서가 정확한가를 결정하는것이다. 심사성원들은 명세서를 검열하면서 문서에 대하여 오해가

없다는것을 담보하게 된다. 관통심사회의와 검토는 두가지 류형의 심사로 되는데 그에 대하여서는 6.2에서 서술한다. 일단 의뢰자가 명세서에 수표하였을 때 진행되는 세부계획작성과 타산에 대한 검열을 생각해 보자. SPMP의 매 측면들이 SQA그룹에 의하여 깐깐히 조사된다는것이 필수적인 반면에 계획의 기한과 가격타산에 특별한 주의를 돌려야 한다. 이를 위한 한가지 방법은 관리자측이 계획작성단계의 초기에 개발기한과 비용이 2개(또는 2이상) 독립적인 타산을 얻고 그다음 중요한 차이를 일치시키는것이다. SPMP와 관련하여 그것을 증명하기 위한 가장 좋은 방법은 명세서의 심사와 류사한 심사를 진행하는것이다. 개발기한과 가격타산이 만족하면 의뢰자는 프로젝트의 개발을 허용한다.

2．3．2．명세작성단계에서의 문서작성

명세작성단계는 두개의 기본결과를 내보낸다. 첫째는 명세서이다. 11장과 12장에서는 어떻게 명세서가 작성되는가를 서술하고 있다. 두번째는 쏘프트웨어프로젝트관리계획이다. SPMP를 설계하는 방법에 대하여 9.3부터 9.5까지에서 주고 있다. 다음단계는 제품을 설계하는것이다.

2．4．설계단계

제품의 명세서는 그 제품이 무엇을 하는가를 알려 준다. 설계단계의 목적은 제품이 그것을 어떻게 수행하는가 하는것을 결정하는것이다. 명세작성을 시작하면서 설계팀은 제품의 내부적인 구조를 결정한다. 설계자들은 제품을 모듈 즉 제품의 나머지부분들에 대한 대면부가 잘 정의되여 있는 독립적인 코드의 부분들로 분해한다(객체는 모듈에 대한 특정한 류형으로 된다.). 매 모듈의 대면부 즉 모듈에 들어 오는 인수들과 모듈로부터 귀환되는 인수들은 상세히 서술되여야 한다. 실례로 어떤 모듈은 핵반응기에서 물의 준위를 측정하며 준위가 너무 낮으면 소리를 내여 경고를 울리게 한다. 군용비행기제품의 객체에 있는 방법은 둘 또는 그이상의 날아 오는 적미싸일의 자리표모임을 입력으로 취하고 그것의 탄도를 계산하며 조종사에게 가능한 회피동작을 알려 주기 위하여 또 다른 객체에 통보문을 보내줄수 있다.

일단 개발팀이 모듈분해(구성방식설계; *architec t ural design*)를 완성하면 상세설계(*detailed design*)가 진행된다. 매 모듈에 대하여 알고리듬이 선택되고 자료구조가 선택된다. 모듈분해가 진행되는 동안 설계팀은 채택되는 설계결정에 대하여 하나의 기록으로 주의 깊게 보존해야 한다. 이 정보는 두가지 리유로 하여 중요하다. 첫째로 제품이 설계되는 동안에 설계팀이 궁지에 빠져 어떤 부분에 되돌아 와 다시 설계를 진행하여야 할 때가 있다. 특정한 결정이 채택된 원인을 기록하면 이러한 정황이 발생하였을 때 개발팀이 그것을 되돌아 추적해 나가는데서 도움이 된다.

설계결정들을 유지해야 할 두번째 리유는 유지정비와 관련되여 있다. 리론적으로 제품설계는 설계전반에 영향을 주지 않으면서 새로운 모듈을 추가하거나 현존 모듈을 교체

하여 앞으로 확장될수 있도록 개방적이여야 한다. 물론 실천적으로 이러한 리상을 실현하기 어렵다. 현실세계에서의 최종기한에 대한 제약은 이후의 설계가들이 확장에 대해서는 생각하지 않고 원래의 명세서를 만족시키는 설계를 최대한 빨리 완성하기 위하여 노력하도록 한다. 이후의 확장(제품이 의뢰자들에게 배포된후에 보충하게 되는)이 명세서에 포함되여 있으면 이것들이 설계에서 고려되여야 하는데 이런 경우는 극히 드물다. 일반적으로 명세서는 이로부터 설계는 현존 요구사항만을 취급한다. 이밖에 제품이 여전히 설계단계에 머물러 있는 동안은 가능한 모든 이후의 확장에 대하여 결정할수는 없다. 마지막으로 설계에서 모든 이후의 가능성을 고려하여야 한다면 결국은 취급하기 힘들게 되며 최악의 경우에는 너무 복잡해서 실현이 불가능하게 된다. 그래서 설계가들은 제품전체를 다시 설계하지 않고 여러가지 합리적인 방법으로 확장될수 있는 설계를 제기하여야 한다. 그러나 주요한 확장을 거치는 제품에서 그 설계가 이후의 변화를 간단히 다룰수 없는 때도 있다. 이러한 단계에 이르면 제품은 전체적으로 다시 설계되여야 한다. 원래의 모든 설계결정들에 대한 근거를 보관한 기록을 리용하여 재설계팀은 보다 쉽게 일할수 있다.

2. 4. 1. 설계단계에서의 시험

2.3.1에서 언급한바와 같이 시험가능성에서 가장 중요한 측면은 추적가능성(*traceability*)이다. 설계의 경우에 이것은 설계의 매 부분이 명세서에 있는 설명문과 련관되여 있다는것을 의미한다. 적당하게 교차참조되는 설계는 SQA그룹에게 설계가 명세서에 부합되는가 그리고 명세서의 매 설명문이 설계의 일정한 부분에 반영되여 있는가를 검열하기 위한 강력한 도구를 제공해 주고 있다.

설계심사는 명세서에서 진행하는 심사와 류사하다. 그러나 대부분의 설계문서의 기술적본성으로 하여 의뢰자는 보통 참가하지 않는다. 설계팀과 SQA그룹성원들은 매 개별적인 모듈은 물론 전반적인 설계를 하면서 설계가 정확하다는것을 담보해야 한다. 찾아야 할 오유류형들에는 론리적오유, 대면부오유, 례외처리(오유조건들의 처리)의 결핍 그리고 가장 중요한것으로서 명세사항에서의 불일치들이 포함된다. 이밖에 심사팀은 일부 명세오유들이 이전단계에서는 발견되지 않을 가능성이 있다는것을 알고 있어야 한다. 심사과정에 대한 상세한 설명은 6.2에서 주었다.

2. 4. 2. 설계단계에서의 문서작성

설계단계의 기본결과물은 설계 그자체인데 그것은 두 부분 즉 모듈에 의한 제품서술인 구성방식설계와 매 모듈에 대한 서술인 상세설계로 이루어 진다. 상세설계는 실현단계를 위하여 프로그람작성자들에게 보내여 진다. 제7장에서는 설계리론을 일반적으로 서술하고 특히는 객체설계를 서술하고 있다. 객체지향설계를 비롯한 설계기법들은 도형 및 의사코드와 같은 설계를 서술하는 방법들과 함께 13장에서 서술하고 있다.

2. 5. 실현단계

실현단계에서 설계를 구성하는 여러가지 구성모듈들이 코드작성된다. 실현단계는 14장과 15장에서 상세히 론의한다.

2. 5. 1. 실현단계에서의 시험

모듈은 실현되는 동안 시험(탁상검열; *desk checking*)되여야 하며 모듈이 실현된 다음에 시험실례에 대하여 실행된다. 이러한 비형식적인 시험은 프로그람작성자들에 의하여 진행된다. 그다음 품질보증그룹이 모듈들을 깐깐히 시험한다. 여러가지 모듈시험기법들은 제14장에서 서술된다. 시험실례들을 실행하는것외에 코드심사는 프로그람작성오유를 발견하는 강력하고 성공적인 기법이다. 여기서 프로그람작성자는 모듈이 렬거된 순서로 심사팀성원들을 안내하게 된다. 심사팀에는 SQA의 대표자가 포함되여야 한다. 처리절차는 앞서 언급한 명세작성과 설계단계의 심사와 류사하다. 다른 모든 단계에서와 같이 SQA 그룹의 활동에 대한 기록을 보관해 두어야 한다.

2. 5. 2. 실현단계의 문서작성

실현단계와 련관된 중요한 문서작성은 적당한 주석을 준 매 모듈에 대한 원천코드이다. 그러나 프로그람작성자들은 코드를 시험하는 모든 시험실례들과 기대되는 결과 및 실제결과들을 비롯하며 유지정비단계를 지원하도록 보충적인 문서작성을 제공한다. 이런 문서들은 2.7.1에서 설명한바와 같이 회귀시험에 리용된다.

2. 6. 통합단계

다음단계는 모듈을 결합하고 제품전체로써 기능이 정확히 수행되는가를 결정하는것이다. 모듈이 통합되는 방법(한번에 모든 모듈을 통합하거나 또는 한번에 하나씩 통합하는 방법)과 특정한 순서(모듈호상련결도에서 하강식, 상승식)는 결과적인 제품의 품질에 결정적인 영향을 줄수 있다. 실례로 제품이 상승식으로 통합된다고 가정하자. 어떤 기본적인 설계오유는 후에 나타나는데 이것은 많은 품을 들여 다시 작성되여야 한다. 반대로 모듈이 하강식으로 통합된다면 보다 낮은 준위의 모듈들은 상승식으로 제품이 통합되는 경우에 하게 되는 시험을 거치지 않게 될것이다. 이것과 기타 다른 문제들에 대하여서는 제15장에서 상세히 서술한다. 실현단계와 통합단계를 병행하여 진행하여야 하는 리유에 대해서는 제2장에서 상세히 서술한다.

2. 6. 1. 통합단계에서의 시험

통합시험(*integration testing*)의 목적은 명세서를 만족시키는 어떤 제품을 만들어 내기 위하여 모듈들이 정확하게 결합되여 있는가를 검열하는것이다. 통합단계의 시험기간

에 모듈의 대면부를 시험하는데 깊은 주의를 돌려야 한다. 중요한것은 형식인수들의 개수와 순서, 형이 실제인수의 개수와 순서, 형과 일치하는것이다. 이런 강력한 형검열[van Wijingaarden et al., 1975]은 콤파일러와 련결프로그람에 의하여 만족스럽게 수행되게 된다. 그러나 많은 언어들은 류사성이 강하지 못하다. 그러한 언어들이 리용될 때에는 대면부검열이 SQA그룹성원들에 의하여 진행되게 된다. 통합단계에서의 시험이 완성되게 되면 SQA그룹은 제품시험을 진행하게 된다. 제품전체로서의 기능검사는 명세서에 준하여 진행되게 된다. 특히 명세서에 명시한 제약들이 시험되여야 한다. 하나의 전형적인 실례는 응답시간이 충분히 짧은가를 시험하는것이다. 제품시험의 목적이 명세가 정확히 실현되였는가를 결정하는것이기때문에 일단 명세서가 완성되면 대부분의 시험실례들이 작성된다. 제품의 정확성뿐만아니라 로바스트성도 시험되여야 한다. 즉 제품이 폭주되는가 또는 제품의 오유처리능력이 나쁜 자료를 처리하는데 적합한가 하는것을 결정하기 위하여 고의적으로 잘못된 오유입력자료들이 제시된다. 만일 제품이 현재 설치된 의뢰자의 쏘프트웨어와 함께 동작하게 되면 시험은 새로운 제품이 의뢰자의 현존 콤퓨터조작에 영향을 주지 않는다는것을 검열할수 있도록 수행되여야 한다. 마지막으로 원천코드와 모든 다른 형태의 문서들이 완전하고 내부적으로 모순이 없는가에 대하여 검열하여야 한다. 제품시험은 15.4에서 론의된다. 통합단계의 시험에서 마지막측면은 인수시험(*acceptance testing*)이다. 쏘프트웨어는 의뢰자에게 배포되는데 의뢰자는 그것들을 실제하드웨어상에서 시험자료와 반대되는 실제자료를 리용하여 그것을 시험한다. 개발팀과 SQA그룹이 아무리 주의를 돌렸다고 해도 본질에 있어서 인공적인 시험실례들과 실제자료사이에는 중요한 차이가 있다. 쏘프트웨어제품은 그 제품이 인수시험을 통과하기전에는 명세서를 만족시킨다고 간주될수 없다. 인수시험에 대해서는 15.5에서 보다 상세히 설명한다.

COTS쏘프트웨어(2.1)의 경우에 제품시험이 완성되자마자 완성된 제품의 판본은 즉석에서 시험하기 위하여 선발된 장래의 의뢰자들에게 제공되게 된다. 이런 첫번째 판본을 알파판본(*alpha version*)이라고 한다. 수정된 알파판본은 베타판본(*beta version*)이라고 한다. 보통 베타판본은 최종판본에 근사하다.

COTS쏘프트웨어에서의 오유는 개발회사에 있어서 제품이 팔리지 않는것으로 하여 막대한 손실을 가져다 준다. 될수록 많은 오유들이 될수록 빨리 발견되도록 하기 위하여 COTS쏘프트웨어개발자들은 자주 즉시시험이 어떤 숨겨 진 오유들을 발견하리라는 기대를 가지고 선발된 회사들에 알파 또는 베타판본을 제공한다. 대신에 알파, 베타측에는 흔히 쏘프트웨어의 배포판에 대한 자유로운 복사가 허용되여 있다. 알파, 베타시험에 참가한 회사에는 위험요소들이 포함된다. 특히 알파시험판본에는 좌절과 시간랑비 그리고 자료기지에 대한 손상을 초래할 오유들이 내재되여 있을수 있다. 그러나 이 회사는 자기의 경쟁자들을 압도할수 있는 새로운 COTS쏘프트웨어를 리용하는데서 첫 시작을 떼였다. 쏘프트웨어기업체들이 SQA그룹에 의한 철저한 제품시험대신에 잠정적인 의뢰자들에 의한 알파시험을 리용할 때 때때로 문제가 발생한다. 비록 서로 다른 많은 장소에서 진행하는 알파시험이 보통 많은 오유들을 발현시킨다고 하여도 SQA그룹이 제공할수 있는 방법론적인 시험을 대신할수는 없다.

2. 6. 2. 통합단계에서의 문서작성

이 단계에서 작성된 문서는 제품전체에 대한 주석을 준 원천코드와 프로젝트전체에 대한 시험실레 그리고 사용자지도서, 조작지도서, 자료기지지도서, 기타 지도서들로 구성된다.

2. 7. 유지정비단계

일단 제품이 의뢰자에게 배포되면 일부 변경시켜야 할 문제들이 제기된다. 유지정비는 제품이 의뢰자의 콤퓨터에 설치된 다음에 마지 못해 수행하는 일이 아니라 초기부터 계획해야 하는 쏘프트웨어개발공정의 필수적인 부분이다. 2.4에서 설명한바와 같이 설계는 실현할수 있는 범위내에서는 이후의 확장을 고려하여야 한다. 코드작성은 장래의 유지정비를 념두에 두고 실행하여야 한다. 결국 2.3에서 강조한바와 같이 유지정비에는 기타 모든 쏘프트웨어활동들을 결합한것보다 더 많은 자금이 소비된다. 그러므로 유지정비는 쏘프트웨어생산에서 매우 중요한 측면으로 된다. 유지정비를 절대로 이후의 문제로 취급해서는 안된다. 대신에 전체적인 쏘프트웨어개발노력은 불가피한 장래의 유지정비의 효과를 최소화하는 방향에서 진행되여야 한다.

유지정비에서 제기되는 공통적인 문제는 문서작성 또는 문서의 결핍이다. 최종기간에 준하여 쏘프트웨어를 개발하는 과정에서 원래의 명세서와 설계문서는 자주 갱신되지는 않으며 그것은 유지정비팀에서는 거의나 리용되지 않는다. 자료기지지도서나 조작지도서와 같은 기타 문서들은 작성되지 않을수도 있다. 왜냐하면 관리자측은 제품을 의뢰자에게 제때에 배포하는것이 쏘프트웨어와 병행하여 문서를 작성하는것보다 더 중요하다고 결정하기때문이다. 많은 실례들에서 원천코드는 유지정비하는 사람에게 제공되는 유일한 문서로 된다. 쏘프트웨어산업에서 자주 직원들이 교체되는것은 유지정비가 실행될 때 원래의 개발자들이 그 기업체에 종사하지 않는다는 점에서 유지정비상황을 악화시킨다.

유지정비는 이미 앞에서 언급한 리유와 16장에서 언급한 보충적인 리유로 하여 쏘프트웨어생산에서 가장 어려운 도전적인 단계로 되고 있다.

2. 7. 1. 유지정비단계에서의 시험

유지정비단계에서 어떤 제품에 대한 변경을 시험하는데는 두가지 측면이 있다. 첫째 측면은 요구하는 변경이 정확히 실현되는가 하는 검열이다. 두번째 측면은 제품에 대하여 요구되는 변경을 실현하는 과정에 다른 우연한 변경이 발생하지 않는다는것을 담보하는것이다. 따라서 일단 프로그람작성자가 요구하는 변경이 실현되였다는것이 확정되면 제품의 나머지 부분의 기능이 손상되지 않았다는것을 확증하기 위하여 제품을 이전의 시험실레로써 시험하여야 한다. 이러한 절차를 회귀시험(*regression testing*)라고 한다. 회귀시험를 지원하기 위하여서는 이전의 모든 시험실레들이 이 시험실레를 실행한 결과와 함께 보존되여야 한다. 유지정비에서의 시험은 16장에서 보다 상세히 고찰된다.

2. 7. 2. 유지정비단계에서의 문서작성

유지정비단계의 하나의 기본측면은 매 변경을 그 변경의 리유와 함께 기록하는것이다. 쏘프트웨어가 변경되면 회귀시험을 진행해야 한다. 따라서 회귀시험실례들은 이 단계에서 문서작성의 기본형식으로 된다.

2. 8. 페 기

쏘프트웨어개발의 마지막단계는 페기이다. 여러해동안 봉사한후 그이상의 유지정비에 더는 비용을 소비할수 없는 단계에 이르게 된다.

1. 때때로 제안된 변경들이 너무 심하므로 전반적인 설계를 변경시켜야 한다. 이러한 경우에 전체적인 프로젝트를 다시 설계하고 다시 코드를 작성하는것은 비용이 더 적게 든다.
2. 원래의 설계를 너무 많이 변경시킨 결과 제품안에 본의 아니게 호상의존성이 발생할수도 있으며 지어는 중요치 않은 하나의 모듈에 대한 자그마한 변경이 제품전반의 기능에 심한 영향을 미칠수도 있다.
3. 문서는 적당하게 유지정비되지 않을수도 있다. 결국 제품을 유지정비하는것보다 또다시 코드작성하는것이 더 안전할 정도로 회귀오유의 위험성이 증가하게 된다.
4. 제품이 실행될 하드웨어(또는 조작체계)가 교체되게 된다. 즉 수정하는것보다도 처음부터 다시 작성하는것이 더 경제적일수도 있다.

이러한 모든 실례들에서 현재의 판본이 새로운 판본으로 교체되며 소트프웨어개발은 계속된다.

다른 한편 진정한 페기는 어떤 제품을 리용할 필요가 없을 때 발생하게 되는 보기드문 사건이다. 의뢰자측은 제품이 제공하는 기능을 더이상 요구하지 않으며 마지막에는 콤퓨터에서 제거되게 된다.

이와 같이 매 단계들에서 주목을 끈 일부 난점들과 함께 완전한 쏘프트웨어개발공정을 검토한 다음 쏘프트웨어생산전반과 관련된 난점들을 고찰해 보자.

2. 9. 쏘프트웨어생산에서의 문제 : 본질적인것과 우연적인것

지난 50여년동안 하드웨어는 보다 값이 눅어 지고 속도가 빨라 졌으며 크기도 줄어들었다. 1950년대에 회사들은 오늘날 1,000딸라미만으로 팔리우고 있는 개인용탁상콤퓨터보다도 능력이 약한 방만한 크기의 콤퓨터를 사는데 이전 가격으로 수백만딸라를 지불하였다. 이러한 경향은 돌려 세울수 없으며 콤퓨터들은 끊임없이 더 작아 지고 더 빨라 지고 값이 더 눅어 질것이다.

그러나 사실은 그렇지 않다. 사실상 많은 물리적인 제약들이 앞으로의 하드웨어의 크기와 속도를 제한하고 있다. 이 제약들중의 첫째가 빛의 속도이다. 전자, 더 정확히 전

자기파는 초당 30만km보다 더 빨리 이동할수는 없다. 그러므로 콤퓨터의 속도를 높이는 한가지 방도는 그것의 구성요소들을 소형화하는것이다. 이런 방식으로 전자들의 이동거리가 보다 짧아 지게 된다. 그러나 구성요소의 크기에 대한 보다 낮은급의 제한도 있다. 전자들은 세개의 원자만한 폭을 가진 좁은 길을 따라서도 이동할수 있다. 그러나 전자들이 이동해야 할 길이 이것보다 더 좁으면 전자들이 어떤 린접한 길로 탈선할수 있다. 같은 리유로 하여 병렬통로들은 서로 너무 가까이 있어서는 안된다. 이렇게 빛의 속도와 령이 아닌 원자의 너비는 하드웨어의 크기와 속도에 물리적인 한계를 준다. 우리는 아직 이러한 한계에 도달하지 못하고 있다. 즉 콤퓨터는 물리적인 한계에 도달함이 없이 적어도 보다 빠르고 보다 작다는 두개의 등급으로 가를수 있다. 그러나 자연의 고유한 법칙으로부터 전자콤퓨터를 임의로 빠르게 또는 임의로 작게 할수는 없다.

그러면 쏘프트웨어의 경우는 어떤가? 쏘프트웨어는 비록 그것이 늘 종이나 디스크와 같은 어떤 물리적인 매체에 기록된다고 하더라도 본질상 개념적이기때문에 비물리적인 실체이다. 얼핏 보기에는 쏘프트웨어로 그 무엇이나 다 할수 있을것 같다. 그러나 흐레드 브룩스는 《은총탄은 없다.》[Brooks, 1986]라는 제목의 기사에서 이러한 믿음을 깨뜨렸다. 그는 하드웨어의 속도와 크기가 물리적으로 실현될수 없는 한계를 가지고 있는것과 류사하게 현재의 쏘프트웨어생산기술로서는 해결할수 없는 본질적인 문제들이 존재한다는것을 밝히였다. 브룩스의 말을 인용하면 다음과 같다. 《쏘프트웨어를 개발하는것은 어려운 일이다. 사실상 은총탄은 없다.》(용어 은총탄에 대한 설명은 아래에 있는 《알고 싶은 문제》를 보시오.)

알고 싶은 문제

브룩스의 기사의 제목에서 용어《은총탄》은 승냥이 같은 사람 즉 다른 말로 승냥이로 돌변한 사람들에 대한 위탁살인방법을 가리키는 말이다. 브룩스의 질문의 방향은 류사한 은총탄이 쏘프트웨어에 대한 문제들을 푸는데 리용될수 있지 않겠는가를 결정하는것이다. 결국 쏘프트웨어는 일반적으로 단순하고 간단한것 같다. 그러나 승냥이 같은 사람과 마찬가지로 쏘프트웨어는 늦어 진 최종기한과 초과된 예산안 그리고 시험에서 발견되지 않은 명세서와 설계에서의 잔류오유와 같은 형태로 그 어떤 무서운 존재로 전환될수 있다.

브룩스의 기사《은총탄은 없다.》를 다시 고찰해 보자.

브룩스의 론점은 쏘프트웨어의 본성으로부터 쏘프트웨어생산에서 제기되는 모든 문제들을 신기하게 해결할수 있는 은총탄은 발견되지 않을것이며 더우기는 은총탄이 하드웨어분야와 견줄만한 돌파구를 쏘프트웨어분야에서 달성하는것을 도울수 없다는것이다. 그는 쏘프트웨어의 난점을 두개의 아리스토텔레스범주로 나누었다. 즉 쏘프트웨어의 고유한 본성으로부터 나오는 고유한 난관을 본질(*essence*)적인 난관으로 그리고 쏘프트웨어생산에서 고유하지 않은 오늘날 우연히 맞다들게 되는 난관들을 우연(*accident*)적인 난관으로 간주하고 있다. 즉 본질적인것은 변화되지 않는 그러한 쏘프트웨어생산측면을 이루고 있는 반면에 우연적인것은 돌파구나 은총탄을 연구하는데 복종해야 한다.

쏘프트웨어생산에서 어느 측면이 고유한 난관으로 되는가? 브룩스는 이에 대하여 네가지 용어 즉 복잡성(*complexity*), 순응성(*conformity*), 가변성(*changeability*), 비가시성(*invisibility*)을 렬거하였다. 브룩스가 복잡성이라는 단어를 리용한것은 그 용어가 일반적으로 콤퓨터과학에서 구체적으로는 쏘프트웨어공학에서 서로 다른 여러가지 의미를 가지게 된다는 점에서 약간 적당치 않다. 브룩스는 《복잡성》이라는 단어를 《복잡한》또는 《착잡한》이라는 의미로 사용하였다. 사실상 이 네가지 측면에 대한 이름들은 비공학적인 의미로 리용되고 있다. 이 네가지 측면들에 대하여 고찰하기로 한다.

2. 9. 1. 복잡성

쏘프트웨어는 인간이 만들어 낸 다른 모든것들보다 더 복잡하다. 하드웨어는 쏘프트웨어에 비하면 거의나 단순한것이다. 이것을 살펴 보기 위하여 콤퓨터의 주기억에서 하나의 16비트 word형자료 W를 고찰하자. 매 비트는 정확히 값 0과 1을 취할수 있기때문에 word형자료 W는 전체적으로 216개의 서로 다른 상태를 가질수 있다. 만일 두개의 16비트 word형자료 W1과 W2가 있다고 하면 W1와 W2가 가질수 있는 가능한 상태의 수는 216의 두제곱인 232개로 된다. 일반적으로 체계가 여러개의 독립적인 부분으로 구성되여 있으면 그러한 체계의 가능한 상태들의 수는 매 구성요소의 가능한 상태수의 적으로 된다. 콤퓨터가 쏘프트웨어제품 P를 동작시키고 16비트 word형자료 W는 옹근수 X의 값을 기억하는데 리용된다고 가정하자. 만일 X의 값이 read(X)라고 하는 명령에 의하여 읽어진다고 하면 옹근수 X는 216개의 서로 다른 값을 가질수 있기때문에 얼핏 보기에는 쏘프트웨어제품이 가질수 있는 상태의 수가 word형자료가 가질수 있는 상태의 수와 꼭같이 보일수 있다. 만일 제품 P가 오직 하나의 명령 read(X)로 구성되여 있으면 P의 상태수는 꼭 216으로 될것이다. 그러나 현실적이고 중요한 쏘프트웨어제품에서 입력으로 되는 변수의 값은 제품에 어디에선가 후에 리용되게 된다. read(X)명령과 X의 값을 리용하게 되는 기타 다른 명령들사이에는 독립성이 존재한다. 만일 제품에서 조종흐름이 X의 값에 의존한다면 정황은 보다 더 복잡해 진다. 실례로 X가 **switch**명령(분기명령)에서 조종변수로 되거나 또는 X의 값에 따라서 끝조건이 결정되는 **for**순환 또는 **while**문이 있을수 있다. 상태수에서 이러한 조합폭발의 결과로 하여 복잡성은 제품의 크기에 비례하여 선형적으로 커지는것이 아니라 훨씬 더 빨리 커지게 된다.

복잡성은 쏘프트웨어의 본질적인 속성이다. 쏘프트웨어의 중요한 부분들이 어떻게 설계되는가에 관계없이 제품의 부분들은 호상작용한다. 실례로 모듈의 상태도 그것의 변수상태에 의존하며 대역변수(하나이상의 모듈에서 호출가능한 변수)들의 상태 또한 전반적인 제품의 상태에 영향을 준다. 실례로 객체지향파라다임을 리용하면 복잡성은 감소될수 있다. 그럼에도 불구하고 복잡성은 전체적으로는 제거될수 없다. 다른 말로 복잡성은 쏘프트웨어의 우연적인 속성은 아니고 본질적인 속성이다.

브룩스는 복잡한 현상이 수학 및 물리학과 같은 학문들에서 서술되고 설명될수 있다고 강조하였다. 수학자들과 물리학자들은 복잡한 체계의 본질적인 특성을 추상화하고 그러한 본질적인 특성만을 반영하는 가장 간단한 모형을 만들고 그 간단한 모형의 타당성을 증명하고 그로부터 예측하는 방법들을 체득하였다. 대비적으로 쏘프트웨어가 단순화

되면 전체적인 련습은 쓸모 없다. 즉 수학과 물리학에서의 단순화기법들은 그러한 체계의 복잡성들이 쏘프트웨어의 경우에서와 마찬가지로 본질적인것이 아니라 우연적인것이기때문에 리용된다.

쏘프트웨어의 이 본질적인 복잡성이 중요한것으로 되는것은 제품을 리해하기가 힘들어 진다는것이다. 사실상 큰 제품을 통채로 리해하는 사람은 없다. 이것은 개발팀성원들사이에 불완전한 통신이 이루어 지게 하며 나아가서 대규모쏘프트웨어생산을 특징 짓는 시간과 비용을 랑비하게 된다. 이밖에 명세에서의 오유는 제품전반에 대한 리해가 부족한것으로 하여 간단히 생기게 된다.

이러한 본질적인 복잡성은 쏘프트웨어공정 그자체뿐아니라 공정관리에도 영향을 준다. 만일 관리자가 관리하는 개발공정에 대하여 정확한 정보를 얻을수 없게 되면 프로젝트의 다음 단계들에서의 직원수요를 결정하고 정확한 예산안을 세우는것이 어렵게 된다. 시간의 흐름과 앞으로의 최종기한과 관련한 고급한 관리에 대한 보고서는 정확치 못할수도 있다. 관리자나 관리자에게 보고하는 사람들이 미결건이 여전히 련관되여야 한다는것을 모르는 경우에 시험일정을 작성하는것은 어려워 진다. 그리고 만일 어떤 프로젝트성원이 떠나갔다면 그를 대신할 사람을 양성하는것이 고통스러울수 있다.

쏘프트웨어의 복잡성에 대한 또 하나의 결론은 그것이 유지정비과정을 복잡하게 한다는것이다. 그림 1-2에서 보여 준바와 같이 전체 쏘프트웨어노력의 약 2/3가 유지정비에 돌려 진다. 유지정비하는 사람들이 제품을 리해하지 못하면 교정유지정비나 확장유지정비는 원래의 유지정비에 의하여 생겨 난 손해를 가시기 위하여 이후의 유지정비가 요구되는것과 마찬가지로 제품에 손해를 줄수 있다. 지어 원래의 저자가 변경을 하였을 때 부주의로 초래되는 이러한 종류의 손해의 가능성은 언제나 존재한다. 그러나 유지정비프로그람작성자가 맹목적으로 작업할 때에는 정황이 더욱 악화된다. 문서가 불충분하거나 더우기 문서가 없거나 문서가 부정확한것은 흔히 유지정비를 정확히 수행하지 못하게 하는 기본원인으로 된다. 그러나 아무리 문서작성이 잘되였다고 하더라도 쏘프트웨어의 고유한 복잡성은 그것에 대처하기 위한 모든 시도들을 초월하고 있다. 즉 이러한 복잡성은 유지정비에 부정적인 영향을 주고 있다. 또한 객체지향파라다임은 복잡성을 감소시킬수 있지만(이로 하여 유지정비를 개선한다.) 그것을 완전히 없앨수는 없다.

2. 9. 2. 순응성

수동적으로 조종되는 금제련소들이 콤퓨터화되게 된다. 일련의 단추들과 조종간에 의하여 제철소를 조종하는 대신에 제철소의 구성부분들에 콤퓨터로부터 필요한 조종신호들을 보낸다. 비록 제련소가 원만하게 운영되고 있다할지라도 경영자들은 콤퓨터화된 조종체계가 금생산을 증대시킬것이라고 생각한다. 쏘프트웨어개발팀의 과제는 현존 기업소와의 대면부를 실현하는 제품을 개발하는것이다. 즉 쏘프트웨어는 기업소에 복종해야 하며 기업소는 쏘프트웨어에 복종하지 않는다. 이것은 브룩스가 정의한 첫번째 류형의 순응성인데 여기서 쏘프트웨어는 현존체계와 대화를 진행하여야 하기때문에 불필요한 복잡성을 초래한다.

새로운 콤퓨터화된 금제련소가 건설되였다면 어쨌단 말인가? 기계기사들과 금속기사,

쏘프트웨어공학자들이 모두 기계와 쏘프트웨어가 본질적이면서도 단순한 방법으로 맞아떨어 지도록 제련소설계를 제의하여야 할것 같다. 그러나 사실상 일반적으로 쏘프트웨어대면부를 다른 요소들에 복종시키는것은 다른 요소들이 이전에 구성된 방식을 변경시키기보다 쉽다는 느낌이 든다. 결국 금제련소의 다른 기사들도 이전과 같이 기계를 설계할것을 주장할것이며 쏘프트웨어는 하드웨어대면부와 일치해야 한다. 이것이 브룩스가 정의한 두번째 류형의 순응성인데 여기서는 쏘프트웨어가 가장 일치한 구성부분이라고 하는 그릇된 견해로 하여 쏘프트웨어는 불필요한 복잡성을 초래하게 된다.

이러한 강제적인 순응성으로 하여 제기되는 문제들은 쏘프트웨어를 재설계하는것으로는 퇴치할수 없다. 왜냐하면 복잡성이 쏘프트웨어 그자체의 구조에 기인되지 않기때문이다. 대신에 그것은 쏘프트웨어설계가들에게 부과된 사람 또는 하드웨어에 대한 대면부에 의해서 초래되는 쏘프트웨어의 구조에 기인된다(앞으로 이것들이 어떻게 변화되는가하는 세부에 대해서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

1.4에 의하면 스타 트랙크 스타쉬프 U.S.S회사에 대한 기술지도서 NCC-17)1-D의 15~17페지에서는 많은 쏘프트웨어생산이 하드웨어가 개발되기전에 시작된다는것을 서술하고 있다[Sternbach and Okura, 1991]. 이러한 관점에서 스타 트랙크가 파학적인 상상이 아니라 과학적인 사실로 되기를 바란다.

2. 9. 3. 가변성

1.1에서 지적한바와 같이 일반기사들에게 다리를 556km 옮기거나 90° 회전시킬것을 요청한다면 리성이 없다고 생각할수 있지만 5년이내에 조작체계의 절반을 다시 만들라고 쏘프트웨어공학자들에게 말하는것은 완전히 타당하다. 일반기사들은 다리의 절반을 다시 설계하는것이 품이 들고 위험하다는것을 알수 있다. 왜냐하면 다리를 처음부터 다시 건설하는것이 비용이 적게 들고 안전하기때문이다. 쏘프트웨어공학자들은 오랜 기간의 실천을 통해 그것을 잘 알고 있다. 그리고 유지정비를 확장하는것이 무분별하며 제품을 처음부터 다시 작성하는것이 흔히 품이 적게 든다는것을 인정할것이다. 그러나 의뢰자들은 흔히 쏘프트웨어에 대한 변경를 요구한다.

브룩스는 쏘프트웨어의 변경이 늘 강요된다는것을 지적하였다. 결국 쏘프트웨어가 동작시키는 하드웨어를 변경시키는것보다도 쏘프트웨어자체를 변경시키는것이 더 쉽다. 즉 그것은 용어 쏘프트웨어가 하드웨어의 리면에 놓여 있는 리유와 관련된다. 이밖에 체계의 기능은 쏘프트웨어에 포함되여 있으며 기법에서의 변경은 쏘프트웨어의 변경으로 실현된다. 빈번하고 철저한 유지정비에 의하여 제기되는 문제들은 단지 무식에 의해서 초래되는 문제들이며 만일 사회전반이 쏘프트웨어의 본질을 더잘 알게 된다면 기본변경에 대한 요구가 발생하지 않을것이라는것이 제기되였다. 그러나 브룩스는 가변성이 쏘프

트웨어의 본질에 대한 속성이며 극복할수 없는 고유한 문제이라는것을 강조하였다. 즉 쏘프트웨어의 중요한 성질은 대중의 교육정도에는 관계없이 쏘프트웨어의 변경은 언제나 강요되며 흔히 이러한 변경들이 철저하게 진행된다는것이다. 쓸모 있는 쏘프트웨어가 변경되여야 할 리유를 다음과 같은 네가지로 말할수 있다.

1. 1.3에서 지적한바와 같이 쏘프트웨어는 현실에 대한 모형이며 현실이 변화될 때 쏘프트웨어는 그에 적응되든가 또는 사멸되여야 한다.
2. 만일 쏘프트웨어가 쓸모 있다고 인식되면 원래설계에 의하여 실현가능한 기능을 초월하여 제품의 기능을 확장하도록 주로 사용자들의 요구를 만족시키는 변경을 할수 있어야 한다.
3. 쏘프트웨어의 가장 큰 한가지 우점은 쏘프트웨어가 하드웨어보다 훨씬 더 변경시키기 쉽다는것이다.
4. 성공적인 쏘프트웨어는 그 쏘프트웨어를 작성한 하드웨어의 생명주기를 초월하여 잘 유지된다. 왜냐하면 흔히 4～5년후에는 하드웨어가 이전처럼 기능을 잘 수행하지 못하기때문이다. 그러나 보다 중요한것은 기술적인 변경이 너무 빨라서 보다 용량이 큰 디스크, 보다 빠른 중앙처리소자 또는 보다 강력한 현시장치와 같은 적당한 하드웨어의 구성부분들이 쏘프트웨어가 존재해 있는 동안은 쓸모가 있다는것이다. 일반적으로 쏘프트웨어는 새로운 하드웨어상에서 실행될수 있을 정도로 변경되여야 한다.

이런 리유로 쏘프트웨어의 한가지 본질은 그것이 변경되여야 한다는것이며 이러한 지속적이고 계속적인 변경은 쏘프트웨어의 품질에 부정적인 영향을 준다.

2. 9. 4. 비가시성

쏘프트웨어의 본질과 관련하여 제기되는 중요한 문제는 그것이 《보이지 않는다는것과 볼수 없다는것》이다. 200페지의 목록을 넘겨 받고 쏘프트웨어를 어떤 방법으로 변경시키라는 과업을 받은 사람은 브룩스의 의도를 정확히 알고 있다. 유감스럽게도 완전한 제품이라든가 그 제품에 대한 어떤 측면의 개관을 표현하기 위한 만족스러운 방도는 없다. 이와 대조적으로 례하면 건축가는 건설해야 할 구조물의 매 세부를 반영하는 2차원설계도와 기타 다른 상세한 도식들뿐아니라 설계전반에 대한 착상을 주는 3차원모형을 제공할수 있다. 화학자들은 분자들의 모형을 만들수 있고 공학자들은 축척모형을 구성할수 있으며 정형외과의사들은 환자들에게 자기들의 얼굴이 외과수술후에 어떻게 보이겠는가를 정확히 보여 주기 위하여 콤퓨터를 리용할수 있다. 선도들은 규소소편과 기타 다른 전자요소들의 구조를 반영하도록 그려 질수 있다. 즉 콤퓨터의 구성요소들은 여러가지 추상화준위에서 여러 종류의 도식들에 의하여 표현될수 있다.

확실히 쏘프트웨어공학자들은 자기들의 제품에 대한 특정한 견해를 표현할수 있는 방법들을 가지고 있다. 실례로 쏘프트웨어공학자는 첫번째는 조종흐름을 보여 주고 두번째는 자료흐름을 보여 주며 세번째는 의존관계의 형태를, 네번째는 시간흐름을 나타내는

방향그라프들을 그릴수 있다. 또한 UML선도(12, 13장)는 대규모쏘프트웨어의 구조를 서술하는 위력한 도구로 인정되였다. 문제는 이러한 그라프들이 거의나 평면적이고 더우기는 계층적이 아니라는것이다. 이러한 그라프들에 있는 많은 교차점들은 리해하는데서 명백히 장애로 된다. 파나스는 하나 또는 그이상의 릉들이 계층구조를 이룰 때까지 그라프의 릉들을 잘라 낼것을 제안하였다[Parnas, 1979]. 문제는 결과적인 그라프가 리해하기 쉽다고 할지라도 그것은 쏘프트웨어의 한 부분만을 볼수 있게 하며 잘라 낸 릉들이 쏘프트웨어의 구성요소들속에 존재하는 호상련관을 리해하는데서 결정적이라는것이다. 이와 같이 쏘프트웨어를 표현할 능력이 결여되여 있는것으로 하여 결국 쏘프트웨어를 리해하기 어려울뿐아니라 쏘프트웨어전문가들사이의 정보전달이 심히 장애된다. 즉 200페지의 목록을 변경해야 할 변경목록과 함께 동업자들에게 넘겨 주기 위한 대안은 없는것 같다.

흐름도표, 자료흐름도(11.3.1), 모듈호상접속도 또는 UML선도(12, 13장)와 같은 모든 종류의 시각화수단들은 제품의 어떤 측면을 시각화하기 위한 매우 유용하고 강력한 수단으로 된다. 시각적인 표현들은 다른 쏘프트웨어공학자들은 물론 의뢰자들과도 정보전달을 진행하기 위한 훌륭한 수단으로 된다. 문제는 이러한 도식들이 제품의 모든 측면들을 구체적으로 표현할수 없으며 제품에 대한 어떤 하나의 시각적인 표현들에서 무엇이 빠졌는가를 결정하는 방법이 없다는것이다.

2. 9. 5. 은총탄은 없는가

브룩스[Brooks, 1986]의 기사가 결코 완전히 막연한것은 아니다. 그는 쏘프트웨어기술에서 세개의 주요한 돌파구가 있다고 서술하였다. 즉 고급언어, 시분할, 쏘프트웨어개발환경(UNIX프로그람작성자들의 작업대(*workbench*)와 같은)이 있다는것을 서술하였지만 그것들은 다만 비본질적이며 우연적인 난관들만을 해결할수 있다고 강조하였다. 그는 정확성증명(6.5), 객체지향설계(13.6), Ada와 인공지능 및 전문가체계를 비롯한 여러가지 기술개발들이 현재 잠재적인 은총탄을 릉가하고 있다고 평가하였다. 비록 이러한 일부 방법들이 우연적인 난점들을 미해결로 남겨 둔다고 하더라도 브룩스는 그것들이 본질적인 난점들과는 상관이 없다고 간주하였다. 앞으로 비교가능한 돌파구를 성취하기 위하여 브룩스는 쏘프트웨어를 개발하는 방법을 바꿀것을 제기하였다. 실례로 가능할 때마다 쏘프트웨어제품들은 주문제품이 아니라 규격제품(즉 COTS쏘프트웨어)으로 판매되여야 한다. 브룩스의 견해에 의하면 쏘프트웨어를 개발하는데서 어려운 부분은 실현단계가 아니라 요구사항확정단계와 명세작성단계, 설계단계에 있다. 그에게 있어서 신속원형작성의 리용은(10.3) 큰 개선을 이룩하기 위한 중요한 원천으로 되고 있다. 브룩스의 제안에서 제품생산성을 크게 개선하기 위한 또 다른 하나의 방안은 증식개발원리를 보다 적극적으로 리용하는것인데 여기서는 제품을 전체적으로 개발할 대신에 계단식으로 개발해 나간다. 이 개념에 대해서는 5.1에서 서술한다.

브룩스는 쏘프트웨어생산을 개선하는데서 주요돌파구를 열기 위해서는 훌륭한 설계자들을 양성하고 고무격려하는것이 필요하다고 하였다. 이미 언급하였지만 브룩스가 제

안한 쏘프트웨어생산의 가장 어려운 측면의 하나는 설계단계이다. 그리고 훌륭한 설계를 작성하려면 능력 있는 설계가들이 있어야 한다. 브룩스는 과거에 인기를 끈 제품들로서 UNIX, APL, Pascal, Modular-2, SmallTalk 그리고 FORTRAN을 렬거하고 있다. 그는 그 쏘프트웨어모두가 한명 또는 몇명의 대가들이 만든 제품들이라는것을 지적하였다. 다른 한편 COBOL, PL/I, ALGOL, MVS/360 그리고 MS-DOS와 같은 보다 단조로우나 쓸모 있는 제품들은 위탁제품들이였다. 브룩스의 견해에 의하면 만일 쏘프트웨어생산을 개선하려고 하는 경우에 설계대가들을 양성하는것이 가장 중요한 목표로 된다.

브룩스의 기사를 읽으면 사기가 떨어 진다. 그는 제목에서부터 현대 쏘프트웨어생산의 고유한 본성(본질)은 은총탄을 찾아 낼 가능성을 미심쩍게 한다는것을 서술하고 있다. 그럼에도 불구하고 그는 만일 신속원형작성과 증식구성기법의 리용 그리고 설계대가를 양성하는것을 통하여 어디서나 살수 있는 기성쏘프트웨어를 사들이는것으로써 쏘프트웨어생산전략을 변경시킨다면 쏘프트웨어의 생산성이 제고될것이라고 결론을 지었다. 그러나 큰 돌파구 즉 《은총탄》은 거의나 없는것 같다. 브룩스의 비관에 대하여 투시적으로 고찰해 볼 필요가 있다. 지난 20년동안 쏘프트웨어산업은 해마다 약 6%정도씩 생산이 확고히 장성하였다는것을 보여 주었다. 이 생산장성은 많은 제조산업들에서 관측된것들과 견줄만하다. 그러나 브룩스가 요구하는것은 《은총탄》 즉 생산성에서 큰 생산장성을 빨리 실현하기 위한 방도이다. 하루밤사이에 두배의 생산성을 기대할수 없다고 하는 그의 견해에 동의하지 않을수 없다. 동시에 6%의 배장성률은 생산성이 12년만에 2배로 된다는것을 의미한다. 이러한 개선은 우리가 만족할만큼 신속하고 극적인것은 못되지만 쏘프트웨어공학과정은 해마다 확고하게 개선되고 있다.

이 장의 나머지부분에서는 공정개선을 목적으로 하는 국가 및 국제적인 시도에 대하여 설명한다.

2．10．쏘프트웨어개발공정의 개선

전반적인 경제는 콤퓨터와 쏘프트웨어에 결정적으로 의존하고 있다. 그러므로 많은 나라들에서 쏘프트웨어개발공정에 관심을 돌리고 있다. 실례로 1987년에 미국방성(DoD; *Department of Defence*)은 다음과 같이 보고하였다. 《새로운 쏘프트웨어방법론들과 기법을 적용함으로써 이루어 지는 생산성과 품질리득에 대하여 아직 리행되지 않는 계약을 체결한 때로부터 20년이 지나서 산업 및 정부기구들은 기본문제가 쏘프트웨어개발공정을 관리할 능력이 없다는것이라는것을 깨닫고 있다.》[DoD, 1987].

이것과 그와 련관된 사건들에 대응하여 DoD는 쏘프트웨어공학협회(SEI; *Software Engineering Institute*)를 창설하고 그것을 경쟁적인 조달과정에 기초하여 피츠버그에 있는 카네기멜론종합대학에 설치하였다. SEI의 하나의 중요한 성공은 능력성숙도모형(CMM)의 시도였다. 쏘프트웨어공정개선과 관련된 노력에는 국제규격화기구의 ISO 9000계렬규격과 ISO/IEC 15504 그리고 40여개 나라를 망라하는 국제적인 쏘프트웨어개선시도들이 포함되여 있다. 우선 능력성숙도모형(CMM)을 서술하자.

2. 11. 능력성숙도모형

SEI의 능력성숙도모형(CMM; *Capability Maturity Model*)은 리용되는 실제의 생명주기 모형과는 상관없이 쏘프트웨어개발공정을 개선하기 위한 전략들의 련관된 그룹이다(용어 《성숙도》는 공정 그자체의 우수성에 대한 척도이다.》. SEI는 쏘프트웨어를 위한 CMM(SW-CMM), 인적자원의 관리를 위한 CMM(P-CMM, 여기서 P는 《사람》을 나타낸다.》, 체계공학을 위한 CMM(SE-CMM), 통합제품개발을 위한 CMM(IPD-CMM) 그리고 쏘프트웨어획득을 위한 CMM(SA-CMM)을 개발하였다. 모형들과 어떤 필연적인 파잉준위사이에는 일련의 불일치가 존재한다. 따라서 1997년에 성숙도모형을 위한 하나의 독립적인 통합틀 즉 능력성숙도모형통합(CMMI)을 개발하기로 결정하였다. 우에 렬거한 5개의 능력성숙도모형가운데서 4개의 모형이 CMMI에 통합되였으며 SA-CMM은 후에 병합되는 것으로 되였다. 추가적인 학문들은 앞으로 보충될수 있다[SEI, 2000]. 지면상의 리유로 하여 여기서는 하나의 능력성숙도모형 SW-CMM만을 설명하기로 한다. SW-CMM은 1986년에 와트홈흐레이가 처음으로 내놓았다[Humphrey, 1989]. 쏘프트웨어개발공정은 활동과 기법 그리고 쏘프트웨어를 개발하는데 리용되는 도구들을 내포하고 있다는것을 상기해 보자. 결국 그것은 쏘프트웨어생산의 기술적인 측면과 관리적인 측면을 다 병합하고 있다. SW-CMM의 기초를 이루고 있는것은 쏘프트웨어개발공정을 관리해 나가는 방법에 의하여 문제들이 초래되기때문에 새로운 쏘프트웨어기법의 리용이 그자체로서는 생산성과 수익성을 증가시키는 결과를 초래하지 않는다는 믿음이다. SW-CMM전략은 기술적인 개선이 어떤 본질적인 결과로 될것이라고 믿고 쏘프트웨어개발공정관리를 개선하게 된다. 개발공정전반에서의 개선은 품질이 보다 좋은 쏘프트웨어를 생성하며 또 시간과 가격이 초과되는것으로 하여 애를 먹게 되는 몇개의 쏘프트웨어프로젝트를 산생시키게 된다.

쏘프트웨어개발공정의 개선이 하루밤사이에 일어 날수 없다는것을 념두에 두면서 SW-CMM은 변경이 증가적으로 진행되도록 한다. 보다 명백하게 5개의 각이한 성숙준위가 정의되며 어떤 개발기업체는 개발공정의 보다 높은 준위를 향하여 일련의 작은 진화단계를 거쳐 서서히 전진하다. 이러한 접근방법에 대한 리해를 돕기 위하여 5개의 준위에 대하여 서술한다.

성숙준위 1. 초기준위 이것은 가장 낮은 준위로서 본질적으로 건전한 쏘프트웨어공학에 대한 관리실천은 기업체에 있다는것이 아니다. 대신에 모든것이 림시적인 기초우에서 진행된다. 유능한 관리자에 의하여 지휘되는 특정한 프로젝트와 훌륭한 쏘프트웨어개발팀은 성공적이다. 그러나 보통의 패턴은 일반적으로는 건전한 관리가 그리고 특수하게는 계획작성의 결여로 인하여 생기게 되는 시간과 가격의 파잉초과이다. 결과 가장 좋은 활동은 미리 계획된 과제가 아니라 중대한 국면들에 대한 대응들이다. 1준위에 있는 기업체들에서 쏘프트웨어개발공정은 예측할수 없다. 왜냐하면 그것이 현재의 지휘자에 전적으로 의존하기때문이다. 즉 지휘자가 변경될 때 그에 따라 개발공정도 변화된다. 결과 제품을 개발하는데 걸리는 시간이라든가 또는 비용과 같은 중요한 항목들을 정확히 예측할수 없게 된다.

유감스러운것은 세계적으로 대부분의 쏘프트웨어기업체들은 모두가 아직 1준위에 해

당한 기업체들이라는것이다.

성숙준위 2. 반복준위 이 준위에서는 기초적인 쏘프트웨어프로젝트관리실천이 진행되게 된다. 계획작성과 관리기법들은 류사한 제품에 대한 경험에 기초한다. 따라서 그 이름을 반복(*repeatable*)준위라고 하였다. 2준위에서는 적당한 개발공정을 달성하는데서 본질적인 첫 단계인 측정이 진행된다. 전형적인 측정은 비용과 일정계획을 주의 깊게 추적하는것을 포함하게 된다. 1준위에서와 같은 위험한 방식으로 기법을 수행하는 대신에 관리자들은 문제가 발생할 때 문제를 식별하고 닥쳐 올 위험을 막기 위하여 즉시적인 정정작용을 취하게 된다. 중요한 점은 측정을 하지 않고서는 그들이 그 문제에서 손을 떼기전에는 그러한 문제들을 발견할수 없다는것이다. 또한 하나의 프로젝트기간에 취해 진 측정은 앞으로의 프로젝트를 위한 실제적인 기한과 비용에 대한 일정을 작성하는데 리용될수 있다.

성숙준위 3. 정의준위 3준위에서 쏘프트웨어제품개발을 위한 공정은 완전히 문서화된다. 개발공정에 대한 관리적인 측면과 기술적인 측면들이 모두 명백히 정의되고 가능한 모든 곳에서 공정을 개선하기 위한 노력이 계속 진행된다. 검토(6.2)는 쏘프트웨어의 품질을 달성하는데 리용된다. 이 준위에서는 앞으로 품질과 생산성을 높이기 위하여 CASE 개발환경(5.6)과 같은 새로운 기술을 도입하는것이 의의 있다. 대조적으로 《고도기술》은 위험구동 1준위공정을 보다 더 혼돈되게 만든다.

비록 많은 기업체들에서 성숙준위 2와 성숙준위 3에 도달하였다고 할지라도 몇개 기업체들만이 성숙준위 4와 성숙준위 5에 이르고 있다. 그렇기때문에 이 두개의 가장 높은 준위는 앞으로의 목표로 된다.

성숙준위 4. 관리준위 4준위의 기업체들은 매개의 프로젝트에 대해서 품질과 생산성에 대한 목표을 설정한다. 이 두가지 량은 련속적으로 측정되며 허용할수 없는 목표로부터의 리탈이 존재할 때 정정작용이 취해 진다. 통계적품질조종[Deming, 1986; Juran, 1988]은 관리자측이 품질이나 생산표준에 대한 중요한 위반으로부터의 탈선을 구별할수 있도록 하는데서 적합하다(통계적인 품질조종의 크기에 대한 실례는 1,000행이 되는 프로그람코드에서 발견되는 오유의 수로 된다.).

성숙준위 5. 최량화준위 5준위에 해당되는 기업체들의 목표는 련속적인 공정개선이다. 통계적인 품질과 공정조종기법들은 기업체를 인도하는데 리용된다. 매 프로젝트로부터 얻게 되는 지식은 앞으로의 프로젝트에서 리용되게 된다. 이리하여 개발공정은 정의 반결합고리를 병합하여 생산성과 품질에 있어서 확고한 개선을 가져 오도록 한다.

5. 최량화준위	공정조종
4.관리준위	공정측정
3.정의준위	공정정의
2.반복준위	기초프로젝트관리
1.초기준위	특정한 공정

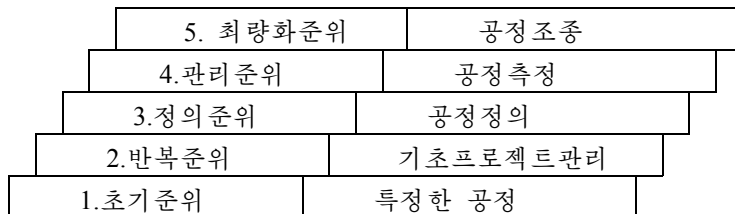



그림 2-1. 능력성숙도모형의 5개의 준위

이러한 5개의 성숙준위들에 대하여 그림 2-1에 개괄하여 보여 준다.

쏘프트웨어개발공정을 개선하기 위하여 기업체는 우선 현재의 개발공정을 리해해야 하며 그다음 앞으로의 개발공정을 형식화하기 위하여 시도한다. 그다음 이러한 개발공정 개선을 실현하기 위한 작용들이 결정되고 우선권순위로 정렬된다. 마지막으로 이러한 개선을 실험하기 위한 계획이 작성되고 실행된다. 개발기업체가 그다음 쏘프트웨어개발공정을 련속적으로 개선해 나아감에 따라 이러한 단계들이 되풀이된다. 즉 이와 같은 높은 준위에로의 진보과정을 그림 2-1에 보여 주었다. 능력성숙도모형으로부터 얻은 경험은 성숙준위를 완전히 달성하는데 보통 18개월부터 3년이 걸리나 1준위에서 2준위로 이행하는데는 때로는 3년 또는 지어 5년까지 걸린다는것을 보여 주었다. 이것은 지금까지 순전히 현재까지 도달한 기초 또는 반작용적인 기초우에서 기능을 수행하여 온 어떤 기업체에 조직적인 방법을 점차적으로 도입하는것이 하는것이 얼마나 어려운것인가를 반영하고 있다.

매개의 성숙준위에 대하여 SEI는 개발기업체들이 그다음 성숙준위에 도달하기 위하여 자기의 노력으로써 다음의 성숙준위에 도달하기 위한 일련의 중요한 개발공정령역들을 강조하고 있다. 실례로 2준위(반복준위)에서의 중요한 공정령역에는 구성조종(5.8)과 쏘프트웨어품질보증(6.1.1), 프로젝트계획작성(9장), 프로젝트추적(9.2.5) 그리고 요구사항관리(10장)가 포함된다. 이러한 령역들은 쏘프트웨어관리의 기초적인 요소들을 포함한다. 즉 의뢰자의 요구(요구사항관리)를 결정하는것, 계획을 작성하는것(프로젝트계획작성), 이 계획으로부터 리탈을 감시하는것(프로젝트추적), 쏘프트웨어제품을 구성하는 여러가지 요소들을 조종하는것(구성관리) 그리고 제품에 오유가 없다는것을 담보하는것(품질보증)들이 포함된다. 매개의 중요한 공정령역내에는 달성되는 경우에 도달되는 다음의 성숙준위를 생성하는 2~4개의 련관목표들의 그룹이 존재한다. 실례로 하나의 프로젝트계획작성목표는 쏘프트웨어개발활동을 적당하게 현실적으로 포함하는 계획의 개발이다.

가장 높은 준위인 성숙준위 5에서 중요한 공정령역들에는 결함방지, 기술혁신공정변화관리가 포함된다. 두개 준위의 중요한 공정령역들을 비교하여 보면 5준위기업체들은 2준위기업체들보다 훨씬 앞섰다는것이 명백하다. 실례로 2준위기업체는 오유를 발견하고 정정하는 (쏘프트웨어품질은 6장에서 보다 더 자세히 론의된다.). 쏘프트웨어품질보증과 관련된다. 이와 대비적으로 5준위기업체들에서 진행하는 공정은 결함방지 즉 쏘프트웨어안에 첫째로 오유가 없다는것을 담보하기 위한 시도를 포함하고 있다. 기업체들이 보다 더 높은 성숙준위에 이르도록 하기 위하여 SEI은 어떤 SEI팀의 평가를 위한 기초로 되는 일련의 질문철을 개발하였다. 평가의 목적은 해당 기업체의 쏘프트웨어개발공정에서 현재의 결함들을 강조하고 기업체가 그러한 공정을 개선할수 있는 방도를 제시하여 주는것이다. 쏘프트웨어공학협회의 CMM계획은 미국방성에 의하여 발기되였다. CMM계획의 원래의 목적의 하나는 국방성쏘프트웨어를 생산하는 계약자들의 공정을 평가하고 어떤 완성공정을 보여 주는 이 계약자들에 대한 계약을 중재함으로써 방어쏘프트웨어의 질을 높이는것이였다. 미공군은 자기의 계약대방으로 되려고 하는 모든 쏘프트웨어개발기업체들은 1998년까지의 SM-CMM 3준위를 준수하여야 한다는것을 규정하였으며 뒤이어 국방성전체가 류사한 지침을 시달하였다. 이리하여 개발기업체들은 자기들의 쏘프트웨어개발공정을 완성하기 위한 개선시도에서 압력을 받게 되였다. 그러나 SM-CMM계획은 국방성쏘

프트웨어를 개선하기 위한 제한된 목표를 훨씬 넘어 섰으며 쏘프트웨어품질과 생산성을 개선하려고 하는 광범한 쏘프트웨어개발기업체들에 의하여 실현되고 있다.

2. 1 2. 기타 쏘프트웨어개발공정개선시도

쏘프트웨어품질을 개선하기 위한 다른 시도는 국제적인 규격화기구(ISO) 9000계렬 표준에 기초하고 있다. 이 계렬의 표준은 설계, 개발, 생산, 설치, 봉사를 비롯한 광범한 산업활동들에 리용될수 있는 계렬화된 5개의 련관된 표준들이다. 즉 ISO 9000이 명백히 쏘프트웨어표준은 아니다. ISO 9000계렬안에서 품질체계를 위한 ISO 9001규격[ISO 9001, 1987]은 쏘프트웨어개발에 가장 널리 응용할수 있는 규격으로 되여 있다. ISO 9001의 보편성으로 하여 ISO는 ISO 9001을 쏘프트웨어에 적용할 때 도움을 주는 특정한 지침 즉 ISO 9000-3[ISO 9000-3, 1991]을 출판하였다.

ISO 9000은 CMM과 구별해 볼수 있는 많은 특징들을 가지고 있다[Dawood, 1994]. ISO 9000은 순응성과 리해성을 담보하기 위하여 설명과 그림으로써 공정을 문서화하는데 중심을 두고 있다. 또한 ISO 9000에 관통되여 있는 원리는 규격의 사항을 준수하는것이 질 높은 제품을 담보하지는 않지만 불충분한 질을 가진 제품이 개발될 위험성을 줄인다는것이다. ISO 9000은 품질보증체계의 일부분일 따름이다. 또한 요구되는것은 품질에 대한 관리자측의 책임, 로동자들의 집중적인 숙련, 련속적인 품질개선을 위한 목표의 설정과 달성이다.

ISO 9000계렬규격들은 미국과 일본, 카나다, 유럽동맹나라들을 포함하여 60개를 넘는 나라들에서 이미 쓰이고 있다. 이것은 실례로 만일 미국의 쏘프트웨어개발기업체가 유럽의 의뢰자와 함께 업무를 진행하려고 하면 미국의 기업체가 우선 ISO 9000의 승인자라는것이 보증되여야 한다는것을 의미하고 있다. 확증된 등록원(검사원)은 회사의 공정을 조사하고 그 공정이 ISO규격에 따른다는것을 보증해야 한다.

유럽의 대방들을 따라서 점점 더 많은 미국의 기업체들이 ISO 9000에 대한 보증을 요구하고 있다. 실례로 제너럴 엘렉트리크 플래스티크 디비젼(General Electric Plastic Division)은 1993년 6월에 340개의 자동판매기가 이 기준에 도달하였다고 주장하였다[Dawood, 1994]. 미행정부가 유럽동맹을 따라 자기 나라에 있는 기업체들과 영업거래를 하려는 다른 나라의 회사들에 대해서는 ISO 9000승인을 요구하는것 같지는 않다. 그럼에도 불구하고 자국내에서와 그 기본무역거래상대들로부터의 압력은 결국 세계적범위에서의 ISO 9000승인을 초래할수도 있다.

ISO/IEC 15504는 ISO 9000과 같이 국제적인 공정개선시도이다. 이러한 시도는 이전에는 쏘프트웨어공정개선능력결정의 략어인 SPICE로 알려 져 있었다. 40여개를 넘는 나라들에서 SPICE를 시도하기 위하여 적극 노력하고 있다. SPICE는 그것을 하나의 국제적인 표준으로서 확립하려는 장기적인 목적과 함께 영국국방성(MOD)에 의하여 시도되였다. SPICE의 첫번째 판본은 1995년에 완성되였으며 1997년 7월에 SPICE의 시도는 국제규격화기구와 국제전기기술의 공동위원회에 의하여 실현되였다. 이러한 리유로 하여 시도의 이름도 SPICE로부터 ISO/IEC15504 또는 간단히 15504로 바뀌여 졌다.

2. 1 3. 쏘프트웨어개발공정개선의 비용과 리득

쏘프트웨어공정을 개선하는것이 수익성을 증가시키는가? 예비적인 결과들은 이것이 정말로 사실이라는것을 보여 주었다. 실례로 캘리포니아의 풀레톤에 있는 후그스항공기의 쏘프트웨어공학기술부에서는 1987년과 1990년사이에 평가개선계획에 약 50만딸라를 소비하였다[Humphrey, Snider and Willis, 1991]. 이 3년기간에 후그스항공기는 앞으로 4준위로 지어는 5준위로 개선되리라는 기대를 가지고 성숙준위 2에서 성숙준위 3으로 이행하였다. 개발공정이 개선된 결과 후그스항공기는 년간절약액이 200만딸라로 추정되였다. 이러한 절약은 여러가지 방법으로 축적되게 되였는데 이 방법들에는 감소된 초과시간, 보다 적은 위험성, 개선된 종업원들의 사기 그리고 쏘프트웨어전문가들의 보다 적은 이동들이 포함된다.

비교할만한 결과들이 기타 다른 기업체들에서 보고되였다. 실례로 레이씨온에 있는 설비디비젼은 1988년의 1준위로부터 1993년에는 3준위로 올라 갔다. 생산성은 두배로 증가되게 되였고 개발공정개선노력에 투자된 비용에 대하여 1딸라당 7.7딸라의 리윤을 얻게 되였다. 쉘름베르그는 류사한 성과를 거두었다[Wohlwend and Rosenbaun, 1993]. 이러한 결과로 능력성숙도모형은 여러 나라의 쏘프트웨어산업에서 비교적 널리 적용되고 있다. 그림 2-2는 초기의 CMM연구에 참가했던 13개의 큰 기업체들에 대한 SEI보고들로부터 뽑은것이다[Herbsleb et al., 1994].

범　주	범위	중간값	자료점의 개수
쏘프트웨어공정개선(SPI)년들	1－9	3.5	24
쏘프트웨어공학자 한사람당 SPI년간비용	490－2,004딸라	1,375딸라	5
년간 생산증가률	9－67%	35%	4
년간 초기오유발견률	6－25%	22%	3
년간 시간에 따르는 판매감소률	15－23%	19%	2
년간 후에 해결된 오유보고서감소률	10－94%	39%	5
영업비(절약 / SPI비용)	4.0－8.8:1	5.0:1	5

그림 2-2. SW_CMM쏘프트웨어개선자료[Herbsleb et al., 1994]
(《CMM지향쏘프트웨어공정개선의 리익성: 초기보고서》에서 표 2를 재작성한
특별허가는 카네기멜론종합대학 1994년판 CMM/SEI-94-TR-013이
쏘프트웨어공학협회에서 승인되였다.)

이 론문에서 인용한 표본은 국방성계약자, 군사기구들 그리고 상업쏘프트웨어기업체들을 비롯한 넓은 범위의 쏘프트웨어개발자들로 구성되였다. 매개 기업체들은 서로 다른 방법으로 비용과 생산성과 같은 량적지표들을 측정하고 이러한 량적지표들이 시간이 감에 따라 어떻게 변화하는가에 초점을 두고 연구를 진행하였다. 그림에서 보여 준바와 같이 생산성과 초기결함발견이 증가하였으며 결함보고서를 조사하고 발표하는 시간은 감소되였다. 그림 2-2는 또한 쏘프트웨어개발공정개선에 지출된 딸라당 절약이 5딸라를 중심

으로 4딸라부터 8.8딸라범위에 있다는것을 보여 주고 있다. 연구를 진행한 13개 기업체들은 SEI SM-CMM은 아닌 여러가지 개발공정개선사업에 적극 참가하였다.

모토롤라정부전자디비젼(GED)은 1992년부터 SEI의 쏘프트웨어개발공정개선계획에 적극적으로 개입하였다[Diaz and Sligo, 1997]. 그림 2-3은 GED프로젝트를 보여 주고 있는데 그것은 매개 프로젝트를 개발한 그룹의 성숙준위에 따라 분류되여 있다. 그림에서 볼수 있는바와 같이 련관된 기간(즉 1992년전에 완성된 기준선에 상대적인 프로젝트의 기간)은 성숙준위의 증가와 함께 감소되였다. 품질은 백만개의 등가적인 아쎔블리어원천행(MEASL)당 오유에 의하여 측정되였다. 즉 서로 다른 언어로 실현한 프로젝트들을 비교할수 있도록 원천코드의 행수는 등가적인 아쎔블리어코드의 행수로 변환된다[Jones, 1996].

CMM준위	기간에 대한 프로젝트수	개발기간에 발견된 상대적인 감소량	상대적인 MEASL당 오유	생산성
준위1	3	1.0	-	-
준위2	9	3.2	890	1.0
준위3	5	2.7	411	0.8
준위4	8	5.0	205	2.3
준위5	9	7.8	126	2.8

그림 2-3. 34개의 모토롤라 GED 프로젝트의 결과

(MEASL는 《백만개의 아쎔블리어원천행》으로 작성되였다.)

［Diaz and Sligo，1997］（©1997，IEEE.）

그림 2-3에서 볼수 있는바와 같이 품질은 성숙준위가 높아 지는데 따라 높아 진다. 마지막으로 생산성은 시간당 사람이 작성한 MEASL로 측정된다. 비밀보장을 리유로 모토롤라는 실제 생산성에 대한 그림을 공개하지 않았다. 그러므로 그림 2-3은 2준위프로젝트의 생산성에 관련되는 생산성을 반영하고 있다(품질이나 생산성에 대한 그림을 1준위프로젝트들에 대해서는 리용할수 없다. 왜냐하면 개발팀이 1준위에 있을 때에는 이러한 량들을 측정할수 없기때문이다.).

이 절에서 서술된것과 이 장의 보충부분에서 렬거된것과 같은 공개된 연구결과들로써 세계적으로 더욱더 많은 기업체들이 개발공정개선이 비용에서 효과적이라는것을 깨닫고 있다.

개발공정개선움직임의 흥미 있는 한가지 효과는 쏘프트웨어개발공정시도와 쏘프트웨어공학표준사이의 호상작용이다. 실례로 1995년에 있은 국제표준화기구는 완전한 쏘프트웨어생명주기에 대한 규격으로서 ISO/IEC 12207을 발표하였다[ISO/IEC 12207, 1995]. 3년후에 규격 IEEE/EIA 12207의 판본이 국제전기 및 전자공학협회(IEEE)와 전자산업동맹(EIA)에 의하여 발표되였다. 이 판본은 많은 쏘프트웨어의 《가장 좋은 실천》을 병합하였다. IEEE/EIA의 승인을 받기 위하여서는 기업체가 CMM능력준위 3근방에 있어야 한다[Ferguson and Sheard, 1998]. 또한 ISO 9000-3은 지금 ISO/IEC 12207의 부분적인것들을 병

합하고 있다. 이것은 쏘프트웨어공학규격화기구와 쏘프트웨어개발공정개선시도사이에서 벌어 지는 이러한 호상작용을 반드시 보다 더 좋은 쏘프트웨어개발공정에로 이끌어 갈것이다.

요 약

몇개의 예비적인 정의를 한 다음(2.1) 요구사항확정단계(2.2)로부터 페기(2.8)에 이르는 쏘프트웨어공정의 매 단계를 서술하였다. 매 단계들과 련관된 문제들에 특별한 주의를 돌렸다. 시험은 모든 쏘프트웨어생산활동과 병렬로 수행되여야 하기때문에 개별적인 단계로 고찰하지 않았다. 매 생명주기단계에서 진행되는 시험에 대한 설명은 2.2.1부터 2.7.1까지에서 주었다. 류사하게 문서작성도 2.2.2에서 2.7.2까지에서 서술한바와 같이 쏘프트웨어제품개발의 고유한 부분이다. 매 개별적인 쏘프트웨어생명주기와 련관된 문제들외에 쏘프트웨어제품개발에서의 본질적인 난점들과 관련한 브룩스의 견해도 서술하였다(2.9). 브룩스가 고찰한 쏘프트웨어의 본질적인 네가지 측면에서의 난점은 복잡성, 순응성, 가변성, 비가시성인데 2.9.1부터 2.9.4에서 서술하고 론의하였다. 브룩스의 견해는 2.9.5에서 분석되였다.

이 장의 마지막부분은 쏘프트웨어개발공정개선이다(2.10). CMMI(2.1)와 ISO 9000과 ISO/IEC 15504(2.12)를 비롯한 국제적인 쏘프트웨어개선시도들을 상세하게 주었다. 쏘프트웨어개발공정개선에 대한 비용의 효과성은 2.13에서 론의하였다.

보 충

제1장의 보충에 있는 검토에 관한 기사들 즉 문헌 [Brooks, 1975; Boehm, 1976; DeMarco and Lister, 1989; Wasserman, 1996; and Ebert, Matsubara, Pezzé, and Bertelsen, 1997]은 또한 쏘프트웨어제품개발과 관련한 문제들을 강조하고 있다.

생명주기의 매 단계에서의 시험과 관련하여 하나의 좋은 참고서는 문헌 [Beizer, 1990]이다. 보다 더 자세한 참고서는 제6장에서 주었고 6장의 보충부분의 마감에 주었다. 여기에서 서술한것과 같은 개요는 브룩스의 기사 《은총탄은 없다.》를 공정하게 평가할수 없다. 이 기사는 원래의 문헌 [Brooks, 1986]에서 읽어야 한다. 브룩스의 기사에 대한 많은 반향들가운데서 콕스는 문헌 [Cox, 1990]에서 사실상 은총탄이 존재한다고 제기하였다. 즉 객체의 재리용이 있다고 제기하였다(8장에서 서술하였다.). 문헌 [Harel, 1992]에서 서술한 견해는 비록 브룩스의 견해와 거의 일치하지만 보다 새로운 기술의 출현과 함께 전망은 브룩스의 견해에서보다는 나쁘지 않다. 문헌 [Mays, 1994]는 은총탄문제에 대한 다른 흥미 있는 견해를 주고 있다. 쏘프트웨어의 복잡성에 대한 론쟁은 *IEEE Computer* 1997년 7월호의 여러 기사들에서 찾아 볼수 있다.

원래의 SEI능력성숙도모형에 대한 상세한 설명은 문헌 [Humphrey, 1989]에서 주고 있다. 능력성숙도모형통합이 문헌 [SEI, 2000]에 서술되여 있다. 문헌 [Humphrey, 1996]은 개인쏘프트웨어개발공정(PSP)를 서술하고 있다. 즉 PSP를 적용한 결과는 문헌 [Ferguson et al., 1997]에 있다. PSP와 관련한 일부 잠재적인 문제들은 문헌 [Johnson and Disney,

1998]에서 론의하였다. PSP와 팀쏘프트웨어개발공정(TSP)는 [Humphrey, 1999]에서 서술되였다. CMM의 성공은 문헌 [Hicks and Card, 1994]에 서술되여 있으며 평가공정에 대한 비평은 문헌 [Bollinger and McGowan, 1991]에 있다. *IEEE Software* 1993년 7월호에는 CMM에 관한 많은 론문들이 포함되여 있다. *IEEE Software* 2000년 7월/8월호에는 쏘프트웨어개발공정완성에 관한 네개의 론문이 있으며 *IEEE Software* 2000년 11월/12월호에는 PSP에 대한 5개의 론문이 있다.

1987년과 1991년사이에 있은 SEI쏘프트웨어개발공정평가에 대한 개관은 문헌 [Kitson and Masters, 1993]에 주었다. 즉 보다 새로운 결과는 문헌 [Herbsleb et al., 1997]에서 통합되였다. 많은 기사들은 SEI개발공정개선계획을 도입한 특정한 회사들의 견지에서 산업적경험에 대하여 서술하고 있다. 즉 전형적인 실례는 쉘렘베르그[Wohlwend and Rosenbaum, 1993]와 레이씨온[Haley, 1996]에서 찾아 볼수 있다. 쏘프트웨어산업에 대한 SEI의 충격은 문헌 [Saiedian and Kuzara, 1995]와 [Brodman and Johnson, 1996]에서 론의되였다. CMM에 대하여 흥미 있는 견해는 문헌 [Bamberger, 1997]에서 보여 주고 있다.

문헌 [Bamford and Deibler, 1993b]는 ISO 9000과 CMM에 대하여 상세한 비교를 주었다. 즉 문헌 [Bamford and Deibler, 1993a]에서 개관을 주었다. 문헌 [Paulk, 1995]에서는 또 다른 하나의 비교를 주었다. 아일랜드의 코르크에 있는 모토롤라그룹이 어떻게 되여 성숙준위 4로 발전하였는가 하는 방법을 문헌 [Fitzgerald and O`Kane, 1999]에서 주었다. CMM에 대한 정보가 풍부하게 지적되여 있는것은 SEI CMM의 web싸이트*) www.sei.cmu.edu이다. ISO/IEC 15504(SPICE) 홈페지는 www.sei.cmu.edu/technology/process/spice/이다.

Proceedings of ACM 1997년 6월호에는 쏘프트웨어의 품질과 CMM에 관한 많은 론문들이 있다. 즉 문헌 [Herbsleb et al., 1997]은 특별히 흥미 있다. 쏘프트웨어공학연구소의 또 하나의 개발공정개선계획은 문헌 [Basili et al., 1995]에 서술되여 있다. 문헌 [Fuggetta and Picco, 1994]은 공정개선에 관한 주석을 단 참고문헌을 주었다. CMM과 IEEE/EIA 12207사이의 비교를 문헌 [Ferguson and Sheard, 1998]에 주었다. 더우기 CMM과 Six Sigma(개발공정개선을 위한 또 하나의 방법)는 [Card, 2000]에서 찾아 볼수 있다. 공정평가에서의 일부 결함들을 문헌 [Fayad and Laitinen, 1997]에 주었다.

문 제

2.1. 의뢰자, 개발자, 사용자들이 다 같은 사람인 상황을 서술하시오.

2.2. 의뢰자, 개발자, 사용자들이 다 같은 사람인 경우에 무슨 문제가 발생하는가? 이 문제들을 어떻게 해결할수 있는가?

2.3. 의뢰자, 개발자, 사용자가 다 같은 사람이면 어떤 잠재적인 우월성이 생기게 되는가?

2.4. 요구사항확정단계와 명세작성단계를 생각해 보자. 그것들을 따로 개별적으로 취

*) 이 책에서 지적한 URL들은 집필시에는 정확한것들이였다. 그러나 web주소는 아주 빈번히 변화되고 있다. 이렇게 되면 독자들은 탐색기구를 리용하여 새로운 URL을 찾아야 한다.

급하는것보다 하나의 단계로 두 공정을 결합하는것이 더 의미가 있는가?

2.5. 쏘프트웨어생명주기의 매 단계들에서 진행되여야 할 문서작성을 렬거하시오.

2.6. 다른 개발단계에서보다 통합단계기간에 그이상의 시험이 진행된다. 이 단계를 두개의 개별적인 단계로 분할하는것이 더 좋겠는가? 즉 하나는 비시험측면을 통합하고 다른 하나는 모든 시험을 통합한다는것이다.

2.7. 유지정비단계는 쏘프트웨어제품개발에서 가장 중요한 단계이며 실현하는데서는 가장 어려운 단계이다. 그럼에도 불구하고 그것은 많은 쏘프트웨어전문가들에게서 차요시되고 유지정비하는 사람들은 다른 개발자들보다 흔히 적은 보수를 받고 있다. 이것이 합리적이라고 생각하는가? 아니라면 그것을 변경하기 위하여 어떻게 시도하겠는가?

2.8. 2.8에서 언급한바와 같이 왜 진실한 페기가 보기 드문 일이라고 하는가?

2.9. 엘메르쏘프트웨어회사에서 화재로 인하여 어떤 프로젝트에 대한 모든 문서가 배포되기전에 소각되였다. 결과 문서작성의 결핍으로 하여 어떤 충격이 있게 되는가?

2.10. 브룩스는 복잡성, 순응성, 가변성, 비가시성으로 하여 쏘프트웨어제품이 실질적으로 어렵다고 하였다. 법률이나 의학, 신학과 공학을 비롯한 인간활동의 다른 령역들도 또한 어렵다. 복잡성, 순응성, 가변성, 비가시성 이것은 이러한 매 령역들에 어느 정도 영향을 주는가?

2.11. 브룩스는 쏘프트웨어기술에서의 가장 중요한 돌파선은 고급언어와 시분할, 쏘프트웨어개발환경이라고 고찰하였다. 가장 주요한 쏘프트웨어돌파선이 무엇이라고 생각하는가? 대답과 함께 그 리유를 설명하시오.

2.12. 교원이 배포한 쏘프트웨어의 상세한 흐름도표를 그리시오. 그것을 리해하기가 쉬운가? 하나의 릉을 차지하는 모든 릉들을 제거함으로써 흐름도표를 평면으로 되도록 하시오. 그 흐름도표가 더 리해하기 쉬운가? 원래의 기능이 얼마나 류실되였는가? 결과적인 평면형흐름도표를 원래의 프로그람작성언어로 반대로 번역하여 실행시키시오. 쏘프트웨어는 《볼수 없으며 볼수 없게 한다.》고 한 브룩스의 진술에 동의하는가?

2.13. 회사가 성숙준위 1에 있기때문에 파산직전에 이른 회사의 쏘프트웨어개발자들을 방금 받아 들였다. 그 회사를 유익하게 되살려 회귀시키려면 처음에 무엇을 해야 하는가?

2.14. (과정안상 목표) 부록 1에 있는 브로드랜즈지역아동병원쏘프트웨어제품에 대하여 쏘프트웨어제품개발에서 본질적인 4개의 문제(복잡성, 순응성, 가변성, 비가시성)이 어떻게 영향을 주는가를 론의하시오.

2.15. (쏘프트웨어공학독본) 교원은 문헌 [Johnson and Disney, 1998]의 복제본을 배포하여 줄것이다. 만일 쏘프트웨어회사를 담당하게 된다면 개인쏘프트웨어개발을 도입하겠는가?

참 고 문 헌

[Bamberger, 1997] J. BAMBERGER, "Essence of the Capability Maturity Model," *IEEE Computer* **30** (June 1997), pp. 112–14.

[Bamford and Deibler, 1993a] R. C. BAMFORD AND W. J. DEIBLER, II, "Comparing, Contrasting ISO 9001 and the SEI Capability Maturity Model," *IEEE Computer* **26** (October 1993), pp. 68–70.

[Bamford and Deibler, 1993b] R. C. BAMFORD AND W. J. DEIBLER, II, "A Detailed Comparison of the SEI Software Maturity Levels and Technology Stages to the Requirements for ISO 9001 Registration," Software Systems Quality Consulting, San Jose, CA, 1993.

[Basili et al., 1995] V. BASILI, M. ZELKOWITZ, F. MCGARRY, J. PAGE, S. WALIGORA, AND R. PAJERSKI, "SEL's Software Process-Improvement Program," *IEEE Software* **12** (November 1995), pp. 83–87.

[Beizer, 1990] B. BEIZER, *Software Testing Techniques,* 2nd ed., Van Nostrand Reinhold, New York, 1990.

[Boehm, 1976] B. W. BOEHM, "Software Engineering," *IEEE Transactions on Computers* **C-25** (December 1976), pp. 1226–41.

[Bollinger and McGowan, 1991] T. BOLLINGER AND C. MCGOWAN, "A Critical Look at Software Capability Evaluations," *IEEE Software* **8** (July 1991), pp. 25–41.

[Brodman and Johnson, 1996] J. G. BRODMAN AND D. JOHNSON, "Return on Investment from Software Process Improvement as Measured by U.S. Industry," *CrossTalk* **9** (April 1996), pp. 23–28.

[Brooks, 1975] F. P. BROOKS, JR., *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley, Reading, MA, 1975. Twentieth Anniversary Edition, Addison-Wesley, Reading, MA, 1995.

[Brooks, 1986] F. P. BROOKS, JR., "No Silver Bullet," in *Information Processing '86,* H.-J. Kugler (Editor), Elsevier North-Holland, New York, 1986. Reprinted in *IEEE Computer* **20** (April 1987), pp. 10–19.

[Card, 2000] D. N. CARD, 'Sorting Out Six Sigma and the CMM," *IEEE Software* **14** (May/June 2000), pp. 11–13.

[Cox, 1990] B. J. COX, "There *Is* a Silver Bullet," *Byte* **15** (October 1990), pp. 209–18.

[Dawood, 1994] M. DAWOOD, "It's Time for ISO 9000," *CrossTalk* (March 1994) pp. 26–28.

[DeMarco and Lister, 1989] T. DEMARCO AND T. LISTER, "Software Development: The State of the Art vs. State of the Practice," *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, May 1989, pp. 271–75.

[Deming, 1986] W. E. DEMING, *Out of the Crisis,* MIT Center for Advanced Engineering Study, Cambridge, MA, 1986.

[Diaz and Sligo, 1997] M. DIAZ AND J. SLIGO, "How Software Process Improvement Helped Motorola," *IEEE Software* **14** (September/October 1997), pp. 75–81.

[Dion, 1993] R. DION, "Process Improvement and the Corporate Balance Sheet," *IEEE Software* **10** (July 1993), pp. 28–35.

[DoD, 1987] "Report of the Defense Science Board Task Force on Military Software," Office of the Under Secretary of Defense for Acquisition, Washington, DC, September 1987.

[Ebert, Matsubara, Pezzé, and Bertelsen, 1997] C. EBERT, T. MATSUBARA, M. PEZZÉ, AND O. W. BERTELSEN, "The Road to Maturity: Navigating between Craft and Science," *IEEE Software* **14** (November/December 1997), pp. 77–88.

[Fayad and Laitinen, 1997] M. E. FAYAD AND M. LAITINEN, "Process Assessment Considered Wasteful," *Communications of the ACM* **40** (November 1997), pp. 125–28.

[Ferguson and Sheard, 1998] J. FERGUSON AND S. SHEARD, "Leveraging Your CMM Efforts for IEEE/EIA 12207," *IEEE Software* **15** (September/October 1998), pp. 23–28.

[Ferguson et al., 1997] P. FERGUSON, W. S. HUMPHREY, S. KHAJENOORI, S. MACKE, AND A. MATVYA, "Results of Applying the Personal Software Process," *IEEE Computer* **30** (May 1997), pp. 24–31.

[Fitzgerald and O'Kane, 1999] B. FITZGERALD AND T. O'KANE, "A Longitudinal Study of Software Process Improvement," *IEEE Software* **16** (May/June 1999), pp. 37–45.

[Fuggetta and Picco, 1994] A. FUGGETTA AND G. P. PICCO, "An Annotated Bibliography on Software Process Improvement," *ACM SIGSOFT Software Engineering Notes* **19** (July 1995), pp. 66–68.

[Haley, 1996] T. J. HALEY, "Raytheon's Experience in Software Process Improvement," *IEEE Software* **13** (November 1996), pp. 33–41.

[Harel, 1992] D. HAREL, "Biting the Silver Bullet," *IEEE Computer* **25** (January 1992), pp. 8–24.

[Herbsleb et al., 1994] J. HERBSLEB, A. CARLETON, J. ROZUM, J. SIEGEL, AND D. ZUBROW, "Benefits of CMM-Based Software Process Improvement: Initial Results," Technical Report CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, August 1994.

[Herbsleb et al., 1997] J. HERBSLEB, D. ZUBROW, D. GOLDENSON, W. HAYES, AND M. PAULK, "Software Quality and the Capability Maturity Model," *Communications of the ACM* **40** (June 1997), pp. 30–40.

[Hicks and Card, 1994] M. HICKS AND D. CARD, "Tales of Process Improvement," *IEEE Software* **11** (January 1994), pp. 114–15.

[Humphrey, 1989] W. S. HUMPHREY, *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989.

[Humphrey, 1996] W. S. HUMPHREY, "Using a Defined and Measured Personal Software Process," *IEEE Software* **13** (May 1996), pp. 77–88.

[Humphrey, 1999] W. S. HUMPHREY, "Pathways to Process Maturity: The Personal Software Process and Team Software Process," *SEI Interactive* **2** (No. 4, December 1999), http://interactive.sei.cmu.edu/Features/1999/June/Background/Background.jun99.htm.

[Humphrey, Snider, and Willis, 1991] W. S. HUMPHREY, T. R. SNIDER, AND R. R. WILLIS, "Software Process Improvement at Hughes Aircraft," *IEEE Software* **8** (July 1991), pp. 11–23.

[IEEE/EIA 12207, 1998] "IEEE/EIA 12207.0-1996 Industry Implementation of International Standard ISO/IEC 12207:1995," Institute of Electrical and Electronic Engineers, Electronic Industries Alliance, New York, 1998.

[ISO 9000-3, 1991] "ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software," International Organization for Standardization, Geneva, 1991.

[ISO 9001, 1987] "ISO 9001, Quality Systems—Model for Quality Assurance in Design/Development, Production, Installation, and Servicing," International Organization for Standardization, Geneva, 1987.

[ISO/IEC 12207, 1995] "ISO/IEC 12207:1995, Information Technology—Software Life-Cycle Processes," International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Johnson and Disney, 1998] P. M. JOHNSON AND A. M. DISNEY, "The Personal Software Process: A Cautionary Tale," *IEEE Software* **15** (November/December 1998), pp. 85–88.

[Jones, 1996] C. JONES, *Applied Software Measurement*, McGraw-Hill, New York, 1996.

[Juran, 1988] J. M. JURAN, *Juran on Planning for Quality*, Macmillan, New York, 1988.

[Kitson and Masters, 1993] D. H. KITSON AND S. M. MASTERS, "An Analysis of SEI Software Process Assessment Results: 1987–1991," *Proceedings of the IEEE 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 68–77.

[Mays, 1994] R. G. MAYS, "Forging a Silver Bullet from the Essence of Software," *IBM Systems Journal* **33** (No. 1, 1994) pp. 20–45.

[Parnas, 1979] D. L. PARNAS, "Designing Software for Ease of Extension and Contraction," *IEEE Transactions on Software Engineering* **SE-5** (March 1979), pp. 128–38.

[Paulk, 1995] M. C. PAULK, "How ISO 9001 Compares with the CMM," *IEEE Software* **12** (January 1995), pp. 74–83.

[Paulk, Weber, Curtis, and Chrissis, 1995] M. C. PAULK, C. V. WEBER, B. CURTIS, AND M. B. CHRISSIS, *The Capability Maturity Model: Guidelines for Improving the Software Process,* Addison-Wesley, Reading, MA, 1995.

[Saiedian and Kuzara, 1995] H. SAIEDIAN AND R. KUZARA, "SEI Capability Maturity Model's Impact on Contractors," *IEEE Computer* **28** (January 1995), pp. 16–26.

[SEI, 2000] "Capability Maturity Model Integration (CMMI): Frequently Asked Questions (FAQ)," Version 1.3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Spring 2000.

[Sternbach and Okuda, 1991] R. STERNBACH AND M. OKUDA, *Star Trek: The Next Generation, Technical Manual,* Pocket Books, New York, 1991.

[van Wijngaarden et al., 1975] A. VAN WIJNGAARDEN, B. J. MAILLOUX, J. E. L. PECK, C. H. A. KOSTER, M. SINTZOFF, C. H. LINDSEY, L. G. L. T. MEERTENS, AND R. G. FISKER, "Revised Report on the Algorithmic Language ALGOL 68," *Acta Informatica* **5** (1975), pp. 1–236.

[Wasserman, 1996] A. I. WASSERMAN, "Toward a Discipline of Software Engineering," *IEEE Software* **13** (November/December 1996), pp. 23–31.

[Wohlwend and Rosenbaum, 1993] H. WOHLWEND AND S. ROSENBAUM, "Software Improvements in an International Company," *Proceedings of the 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 212–20.

제3장. 쏘프트웨어생명주기모형

쏘프트웨어생산은 보통 애매한 개념들로 시작된다. 즉《만일 콤퓨터가 방사능준위에 대한 그라프를 그려 낼수 있다면 그것은 좋은것이 못된다.》또는《만일 이 회사가 매일 기초로 되는 현금의 흐름에 대한 정확한 표상을 가지고 있지 않다면 우리는 여섯달만에 빚을 갚을수 없게 된다.》또는 지어《우리가 이러한 새로운 류형의 표처리프로그람을 개발하여 시장에 내가면 백만딸라를 벌게 될것이다.》라는 등의 애매한 개념들로 시작된다. 일단 제품에 대한 요구가 설정되면 제품은 일련의 개발단계를 거치게 된다. 대표적으로 제품은 명세작성되고 설계되며 실현되게 된다. 만일 의뢰자가 만족하게 되면 제품은 설치되여 동작하는 기간에 유지정비된다. 만일 제품이 이제 더는 쓸모가 없게 되면 그것은 페기된다. 제품이 발전하게 되는 이러한 단계를 생명주기모형(*life-cycle model*)이라고 부른다.

매 제품의 생명주기경력은 서로 다르다. 일부 제품들은 여러해동안 개념적인 단계에 머물러 있게 된다. 왜냐하면 현재의 하드웨어가 그 제품이 충분히 실행될수 있을 정도로 빠르지 못하거나 효률적인 알고리듬이 개발되기전에 기본적인 연구가 진행되여야 하기때문이다. 일부 제품들은 빨리 설계되고 실현된 다음 사용자들의 요구에 맞게 수정되도록 하는 유지정비단계에서 여러해 걸리게 된다. 또한 일부 제품들은 설계되고 실현되고 유지정비된다. 기본적인 유지정비를 몇해 진행한 다음에는 현재의 판본을 다시 보수하려고 하는것보다 완전히 새로운 제품을 개발하는것이 더 값이 눅게 된다.

이 장에서는 여러가지 서로 다른 생명주기모형들에 대하여 서술한다. 가장 널리 리용되는 두가지 모형은 폭포모형과 신속원형작성모형이다. 이밖에 라선모형이 지금에 와서 상당한 주목을 끌고 있다. 이 세 모형들의 우점과 약점을 밝히기 위하여 증식모형과 동기 및 안정화모형 그리고 매우 충분하지 못한 구성 및 수정모형을 비롯한 기타 생명주기모형들도 고찰한다.

3. 1. 구성 및 수정모형

많은 제품들이 구성 및 수정모형을 리용하여 개발된다는것은 유감스러운 일이다. 그 제품은 명세서도 없이 구성되거나 설계에 진입하게 된다. 대신에 개발자들은 의뢰자들을 만족시키기 위하여 필요한 회수만큼 재작업하게 되는 어떤 제품을 개발하게 된다. 이러한 방법을 그림 3-1에 보여 주었다.

비록 이 연구방법이 100 또는 200개행의 크기를 가진 짧은 프로그람을 작성하는 련습에서는 잘 동작할수 있다고 하더라도 구성 및 수정모형은 임의의 합당한 크기를 가진 제품에 대해서는 전혀 충분하지 않다. 만일 그에 대한 변경이 요구사항확정과 명세작성 또는 설계단계에서 이루어 지면 쏘프트웨어제품을 변경시키는데 드는 비용은 상대적으로 작아 지지만 제품이 다 코드작성된 다음에 변경이 이루어 지면 비용은 매우 커지게 되고

또는 이미 유지정비단계에 들어 섰다면 정황은 더 불리해 진다는것을 그림 1-5에서 보여주고 있다. 결국 구성 및 수정방법에 드는 비용은 사실상 적당하게 명세작성되고 주의 깊게 설계된 제품의 비용보다도 훨씬 더 크게 된다. 이밖에 제품에 대한 유지정비는 명세서나 설계문서가 없으면 매우 곤난하게 되며 회귀오유가 일어 날수 있는 기회는 상당히 더 커지게 된다.

그림 3-1. 구성 및 수정모형

구성 및 수정모형대신 제품개발을 시작하기전에 전면적인 방식계획이나 생명주기모형이 선택되는것은 필수적이다. 생명주기모형(때때로 모형으로 생략한다.)은 요구사항확정과 명세작성, 설계, 실현, 통합과 유지정비와 같은 쏘프트웨어개발공정의 여러단계들을 그것들이 진행되는 순서에 따라 규정하고 있다. 일단 모든 당사자들이 생명주기에 동의하면 제품의 개발을 시작할수 있게 된다.

1980년대 초까지 폭포모형은 널리 리용되여 온 유일한 생명주기모형이였다. 이 모형에 대하여 이제 상세히 설명하기로 한다.

3. 2. 폭포모형

폭포모형은 로이스[Royce, 1970]가 처음으로 내놓았다. 이 모형의 한가지 류형을 그림 3-2에 제시한다.

우선 의뢰자와 쏘프트웨어품질보증(SQA)그룹에 의해서 요구사항이 결정되고 검사된다. 그다음 제품에 대한 명세서가 작성된다. 즉 제품이 무엇을 하여야 하는가를 서술하는 문서가 작성된다. 명세서는 SQA그룹에 의해서 검사되고 의뢰자들에 보여 주게 된다. 일단 의뢰자가 명세서에 수표하면 다음단계에서는 쏘프트웨어를 개발하기 위한 상세한 시간표로 되는 쏘프트웨어관리계획이 작성된다. 이 계획도 역시 SQA그룹에 의하여 검사된다. 의뢰자가 개발자의 개발기간과 그 제품에 대한 비용타산에 찬성하면 설계단계가 시

작된다. 명세서가 그 제품이 무엇을 하게 되는가를 설명한다면 설계문서는 그 제품이 그것을 어떻게 하는가 하는것을 서술한다.

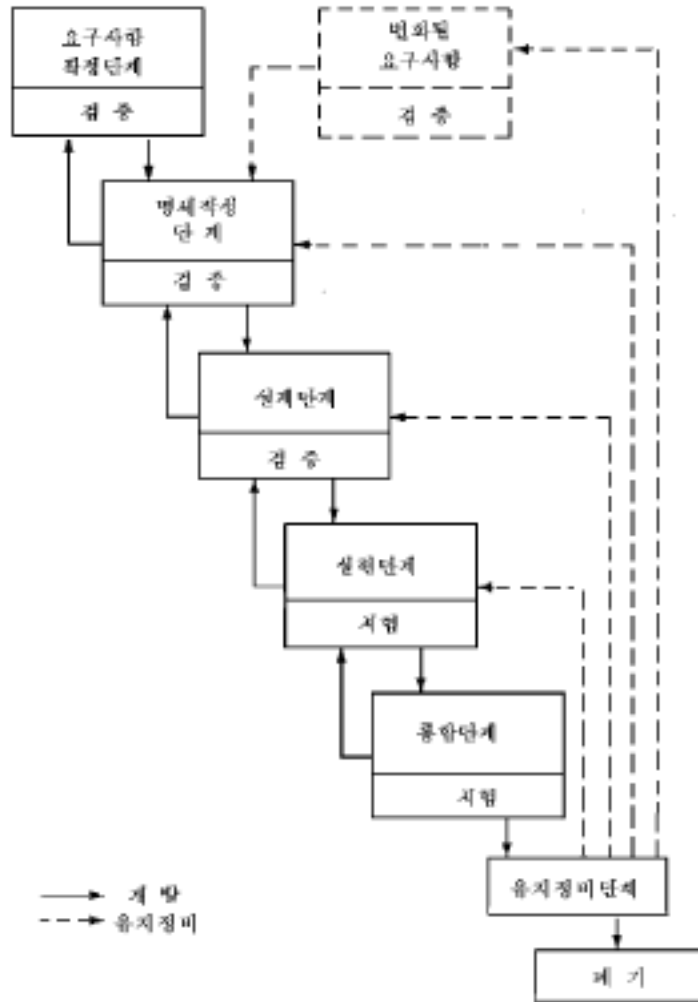



그림 3-2. 폭포모형

설계단계에서는 때때로 명세서에 있는 오유가 명백해 지게 된다. 명세서는 불완전할수도 있고(제품에 대한 일부 특성이 빠져 있다.) 모순적일수도 있고(명세서에서 둘이상의 설명문이 량립될수 없는 방법으로 제품을 정의한다.) 또는 애매할수도 있다(명세서가 하나이상의 가능한 해석을 포함하고 있다.). 불완전성과 모순성, 애매성이 존재하는것은 쏘프트웨어개발공정을 거치기전에 명세서에 대한 교열을 진행할것을 요구한다. 다시 그림 3-2를 참고하면 왼쪽의 설계단계통으로부터 명세작성단계통으로 거꾸로 그어 진 화살표는 반결합고리를 이룬다. 만일 개발자들이 설계단계에서 명세서를 교열하면 쏘프트웨어 생산공정은 이러한 반결합고리를 따라 진행되게 된다. 의뢰자의 허가를 받고 명세서에서

필요한 변경들이 진행되며 계획작성과 설계문서도 이러한 변경을 병합하도록 조정된다. 개발자들이 최종적으로 만족하게 되면 설계문서는 실현을 위하여 프로그람작성자들에게 넘겨 진다. 설계단계에서의 결함은 실현단계에도 반영된다. 어떤 실시간체계에 대한 설계는 실현될 때 아주 느리다고 인정될수 있다. 이러한 설계상 결함에 대한 실례는 대부분의 콤파일러가 어떤 배렬 b의 요소들을 행 우선순서로 즉 b(1.1), b(1, 2), b(1, 3), …, b(1, 17), b(2, 1), b(2, 2), b(2, 3), …, b(2, 7)으로 기억하기 위한 코드를 발생한다는 사실로부터 생겨 난다. 하나의 200×200배렬 b가 블로크의 한 행으로 디스크에 기억된다. 즉 200개의 단어를 가진 행이 read명령이 실행될 때마다 주기억의 완충기에로 읽혀 진다고 가정하자. 만일 배렬이 한 행씩 읽혀 진다면 그때는 모든 40,000개의 원소가 모두 읽혀 지도록 정확히 200개의 블로크가 디스크로부터 주기억에 전송되여야 한다. 첫 read명령문이 첫행을 완충기에 넣어 지도록 하며 첫 200개의 read명령문은 완충기의 내용을 리용하게 된다. 그러면 201번째 원소가 읽어 질 때에만 디스크로부터 주기억에 전송되여야 할 두번째 블로크가 요구되게 된다. 그러나 만일 제품이 배렬을 한 렬씩 읽는다면 서로 다른 행들을 호출하기때문에 매 read명령문에 대하여 새로운 블로크가 전송되여야 한다. 이리하여 배렬이 행 우선순서로 읽혀 질 때의 200개 블로크대신에 40,000개 블로크전송이 요구되며 제품의 그 부분에 대하여 입출력시간은 200배나 더 길어 지게 된다. 이러한 류형의 설계결함은 팀에서 쏘프트웨어개발을 계속할수 있게 되기전에 퇴치되여야 한다. 실현단계에서 그러한 반결합고리를 가진 폭포모형은 설계문서와 명세서, 필요하다면 요구사항에 대하여 변경이 진행되도록 한다.

모듈은 실현되고 문서화되고 통합되여 하나의 완전한 제품을 형성하게 된다(실천적으로 실현과 통합단계는 보통 병렬로 수행된다. 1.5에서 서술한바와 같이 매 모듈은 그것이 실현되고 시험되자마자 통합된다.). 통합단계에서는 명세서와 설계문서에 대한 수정에 앞서 코드를 거꾸로 추적하여 변경을 진행하는것이 필요하다.

폭포모형과 관련한 중요한 점은 그 단계에 대한 문서작성이 완성되고 그 단계에서의 제품이 SQA그룹에 의해서 승인될 때까지 그 어느 단계도 완성되지 않는다는것이다. 이것은 변경을 하여야만 된다. 즉 만일 어떤 앞단계에서의 제품이 그다음에 오는 반결합고리에 따라서 변경되여야 한다면 그 앞선 단계는 그 단계에 대한 문서가 변경되고 SQA그룹에 의해서 그 변경이 검사될 때에만 완성될수 있다고 간주된다. 개발자들이 제품이 성과적으로 완성되였다고 생각되게 되면 제품은 인수시험을 받기 위하여 의뢰자에게 넘겨진다. 이 단계에서 배포물들에는 계약에 렬거된 사용자지도서와 기타 문서들이 포함된다. 제품이 실지로 그 명세서를 만족시킨다고 의뢰자들이 인정할 때 그 제품은 의뢰자에게 넘겨 져서 의뢰자의 콤퓨터에 설치된다. 일단 의뢰자가 제품을 인수하면 잔류오유들을 제거하거나 기타 방법으로 제품을 확장하려고 하는 그러한 모든 변경들이 유지정비를 이룬다. 그림 3-2에서 볼수 있는바와 같이 유지정비는 바로 실현단계의 변경뿐아니라 설계단계와 명세작성단계에서의 변경도 요구하게 된다. 이밖에 확장은 요구사항확정단계에서의 변경에 의하여 유발된다. 이것은 명세서와 설계문서, 코드에서의 변경을 통하여 차례로 실현된다. 폭포모형은 동적모형이며 반결합고리는 이러한 동적구조에서 중요한 역할을 한다. 또한 문서작성이 코드 그자체와 마찬가지로 조심히 유지정비되고 매 단계에서의 제품이 다음단계가 개시되기전에 주의 깊게 검사되는것이 사활적이다. 폭포모형이 리

용되여 폭 넓고 다양한 여러가지 제품들에서 큰 성과를 거두었다. 그러나 실패도 있었다. 어떤 프로젝트에 대한 폭포모형의 리용여부를 결정하기 위해서는 그것의 약점과 우점을 리해하는것이 필요하다.

3. 2. 1. 폭포모형의 분석

폭포모형은 학문적성격이 강한 방법 즉 매 단계에서 문서작성이 진행되고 매 단계에서의 모든 제품들이(문서작성을 포함) SQA에 의해서 주의 깊게 검사되여야 한다고 하는 요구를 비롯하여 많은 우월성을 가지고 있다. 매 단계를 끝내는 리정표에 대한 한가지 본질적인 측면은 그 단계에 대하여 명시된 모든 문서들을 비롯하여 그 단계의 모든 제품들이 SQA그룹에 의하여 증명되여야 한다는것이다. 폭포모형의 매 단계에서 고유한것은 시험이다. 시험단계는 제품이 구성되였을 때에만 진행되는 분리된 단계가 아니며 매 단계의 마감에만 진행되는것은 아니다. 대신에 제2장에서 언급한바와 같이 시험은 쏘프트웨어개발공정전반에 걸쳐 계속 진행되여야 한다. 특히 요구사항확정이 진행되는 동안 그것들은 확증되여야 하며 설계문서는 물론 명세서와 쏘프트웨어관리계획도 확증되여야 한다. 코드는 여러가지 방법으로 시험되여야 한다. 유지정비단계에서는 변경된 제품판본이 여전히 그이전 제품판본이 수행하던것을 할수 있으며 즉 여전히 그것이 정확하게 수행된다는것(회귀시험)뿐아니라 의뢰자에 의해서 강요된 그 어떤 새로운 요구를 전적으로 만족시킨다는것을 담보해야 한다.

명세서와 설계문서, 코드문서와 그리고 자료기지도서, 사용자지도서, 조작지도서와 같은 기타 문서들은 제품을 유지정비하기 위한 필수적인 수단으로 된다. 제1장에서 언급한바와 같이 쏘프트웨어개발예산의 평균 67%는 유지정비에 들며 이러한 문서조항들로써 폭포모형을 지지하면 이러한 유지정비가 더 쉽게 된다. 앞부분에서 언급한바와 같이 쏘프트웨어생산에 대한 그러한 질서정연한 방법은 유지정비단계에서 계속된다. 매개의 변경은 련관된 문서에 반영되여야 한다. 폭포모형에서의 많은 성과들은 문서구동모형으로서의 그것의 본질에 기인된다.

그러나 폭포모형이 문서구동이라는 사실은 또한 불리한 경우도 있을수 있다. 이것을 보기 위하여 다음 두개의 약간 기묘한 대본에 대하여 고찰해 보자. 우선 죠아와 제인 존슨은 집을 지으려고 결심한다. 그들은 건축설계가와 의논한다. 건축설계가는 그들에게 설계안(스케치)과 계획 그리고 모형을 보여 줄 대신에 고급한 기술용어로 집을 묘사한 20페지나 되는 빼곡히 타자한 간단한 문서들을 준다. 지어 죠아와 제인이 사전 건축설계경험이 없고 문서를 힘겹게 리해한다고 할지라도 그들은 그 문서에 열정적으로 서명을 하고 다음과 같이 말한다. 《곧바로 가서 집을 짓자.》 다른 대본은 다음과 같다. 마크 마베리는 우편주문으로 옷을 산다. 회사는 마크에게 옷도안과 입을만한 옷들의 원형을 우편으로 보낼 대신 그 제품에 대하여 서면작성한 재단과 천에 대한 사항을 보낸다. 마크는 작성된 사항에 기초해서 독단적으로 옷을 주문한다.

우의 두 대본은 아주 다르다. 그럼에도 불구하고 그것들은 폭포모형을 리용하여 쏘프트웨어들을 개발하는 방법을 정확히 특징 짓고 있다. 개발공정은 명세작성으로부터 시작된다. 일반적으로 명세서들은 길고 상세하며 솔직히 말하여 읽기가 지루하다.

의뢰자는 보통 쏘프트웨어명세서를 리해는데서 경험이 없다. 이러한 난점은 명세서가 보통 의뢰자에게 익숙되지 못한 문체로 씌여 졌기때문에 혼돈된다는것이다. 명세서가 Z와 같은 형식적명세작성언어[Spivey, 1992]로 작성될 때 난관은 더욱 커지게 된다(11.8). 그럼에도 불구하고 의뢰자는 적당히 리해했든 못했든 명세서에 수표하고 전진한다. 다만 부분적으로나마 리해되게 작성된 사항으로부터 집을 세우자고 계약한 죠아와 죤슨 그리고 다만 부분적으로 리해한 명세서에 의하여 서술된 쏘프트웨어제품을 승인한 의뢰자들사이에는 많은 측면에서 차이가 거의 없다. 마크 마베리와 그의 우편주문옷은 아주 기묘하지만 그것은 폭포모형이 쏘프트웨어생산에 리용될 때 무슨 일이 벌어 지겠는가 하는것을 명백하게 하여 준다. 의뢰자가 동작하고 있는 제품을 처음으로 보게 되는 때는 전체 제품이 코드작성된 이후뿐이다. 쏘프트웨어개발자들이 《내 알기에는 이것이 내가 요청한것이지만 사실은 내가 바라는것이 아니다.》라는 문장을 무서워 하면서 살고 있다는것은 좀 이상하다.

무엇이 잘못되였는가? 의뢰자들이 명세서에 서술된대로 제품을 리해하는 방법과 실제의 제품을 리해하는 방법사이에는 중요한 차이가 있다. 명세서는 종이우에 존재할뿐이다. 따라서 의뢰자는 그 제품자체가 무엇과 같은가를 실지 리해할수 없다. 작성된 명세서에 크게 의존하는 폭포모형은 의뢰자의 실제요구를 단순히 충족시킬수 없는 제품구성에로 이끌어 갈수 있다. 공정하게 말하면 바로 건축설계가들이 모형이나 설계초안, 계획들을 제공함으로써 의뢰자들로 하여금 무엇을 건설하게 되는가를 리해할수 있도록 하는것과 마찬가지로 쏘프트웨어공학자들도 의뢰자와 통신하기 위하여 자료흐름도(11.3.1)와 같은 도형적인 기법을 리용할수 있다는것이 지적되여야 한다. 문제는 이러한 도형적인 수단으로 완성된 제품이 어떻게 동작하는가를 서술하지 않는다는데 있다. 실례로 흐름도(생산에 대한 도식적인 서술)와 동작하는 제품 그자체사이에는 중요한 차이가 있다. 시험되여야 할 다음 생명주기모형 즉 신속원형작성모형의 우점은 그것이 의뢰자들의 실제요구에 맞는다는것을 담보할수 있도록 해준다는것이다.

3. 3. 신속원형작성모형

신속원형(*rapid prototype*)은 제품의 부분모임과 기능상 등가인 작업모형이다. 실례로 만일 목적하는 제품이 수입계산이나 지불계산, 창고를 관리하는것이라면 신속원형이 자료포착을 위한 화면조종을 진행하고 보고서를 인쇄하는 제품으로 구성될수 있지만 파일갱신과 오유조종을 하지 못하는 그러한 제품으로 구성될수 있다. 용액의 효소농도를 결정하는것을 목적으로 하는 제품의 신속원형은 계산을 진행하고 해답을 현시하는것이지만 입력자료에 대한 그 어떤 타당성이나 합리성검사는 하지 않는다. 그림 3-3에 제시한 신속원형작성생명주기모형에서 첫 걸음은 신속원형을 작성하고 의뢰자와 장래의 사용자들이 신속원형을 리용하여 대화하면서 실험하게 하는것이다. 일단 의뢰자들이 신속원형이 대부분의 실제요구를 실지로 만족시킨다는것을 확인하면 개발자들은 제품이 의뢰자들의 실제요구를 만족시킨다는 일련의 담보를 가지고 명세서를 작성할수 있다. 신속원형작성을 진행하면서 쏘프트웨어개발공정은 그림 3-3과 같이 계속 진행된다.

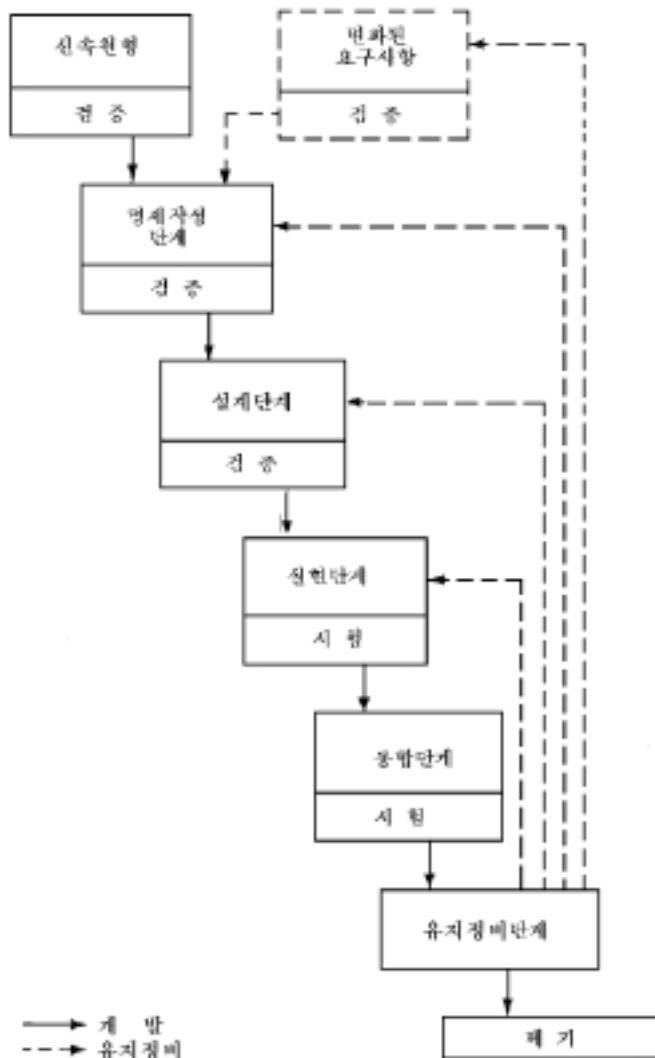



그림 3-3. 신속원형작성모형

신속원형작성모형의 중요한 우점은 제품개발이 본질에 있어서 신속원형작성으로부터 배포제품에 이르기까지 선형으로 진행된다는것이다. 즉 폭포모형(그림 3-2)의 반결합고리들은 신속원형작성모형에서 필요 없을것 같다. 이에 대해서는 여러가지 리유가 있다. 첫째로 개발팀성원들은 명세서를 작성하기 위하여 신속원형을 리용한다. 동작하는 신속원형이 의뢰자들과의 대화를 통하여 타당성이 증명되기때문에 결과적인 명세서가 정확할것이라고 기대하는것은 옳다. 둘째로 설계단계를 고찰하자. 지어 신속원형이(아주 정확하게) 서둘러 작성되였다고 하더라도 설계팀은 그것을 리해할수 있다. 즉 최악의 경우에 그 신속원형들은 《하지 말아야 할 방법》에 대한 변종으로 될것이다. 또한 폭포모형에서의 반결합고리는 여기서 더이상 필요되는것 같지 않다.

다음에는 실현단계가 진행된다. 폭포모형에서 설계의 실현은 때때로 소소하게 제기되는 설계상오유와 맞다들게 된다. 신속원형작성모형에서 초보적으로 동작하는 모형이 이미 작성되였다는 사실은 설계기간에 또는 실현된 다음에 수정에 대한 요구를 줄이게 한다. 원형은 비록 그것이 완전한 목표제품에 대한 부분적인 기능만을 반영할수 있다고 하더라도 설계팀에 일련의 견해을 준다. 일단 의뢰자가 제품을 인수하고 설치하면 유지정비가 시작된다. 진행될 유지정비에 의존하여 요구사항확정단계, 명세작성단계, 설계단계이든가 실현단계에로 순환이 다시 들어 갈수 있다.

신속원형작성의 본질적인 측면은 단어 《신속》에서 구체화된다. 개발자들은 쏘프트웨어개발공정을 다그칠수 있게 될수록 신속하게 원형을 작성하려고 시도한다. 결국 신속원형의 유일한 리용목적은 의뢰자의 실제요구가 무엇인가를 결정하는것이다. 즉 일단 이것이 결정되면 신속원형실현은 무시되지만 얻은 교훈은 남아서 그이후의 개발단계들에서 의연히 리용되게 된다. 이러한 리유로 하여 신속원형의 내부구조는 상관이 없다.

중요한것은 바로 원형을 신속하게 작성하고 의뢰자들의 요구를 반영하여 신속하게 변경되도록 하는것이다. 그러므로 속도가 필수적이다.

3．3．1．폭포모형과 신속원형작성모형의 통합

폭포모형은 성공적임에도 불구하고 의뢰자들에게 배포된것은 실지 의뢰자들이 요구하는것은 아닐수 있다는 약점을 가지고 있다. 신속원형작성모형 역시 많은 성과를 거두었다. 그럼에도 불구하고 제10장에서 서술하는바와 같이 아직도 모든 의심이 가셔지지 않았으며 모형은 그자체의 약점을 가지고 있다.

한가지 해결방도는 두개의 연구방법을 결합하는것이다. 즉 이것은 그림 3-2(폭포모형)에서의 단계들과 그림 3-3(신속원형작성모형)에서의 단계들을 비교하여 고찰할수 있다. 신속원형작성은 요구사항분석기법으로 리용될수 있다. 즉 달리 말하면 첫단계는 의뢰자의 실제요구를 결정하는 신속원형을 작성하는것이고 그 다음단계는 폭포모형에 대한 입력으로서 그 신속원형을 리용하는것이다.

이러한 연구방법은 쓸모 있는 측면효과를 가지고 있다. 일부 개발기업체들은 그 어떤 새로운 기법의 리용이 내재하고 있는 위험으로 인하여 신속원형작성방법을 리용하는것을 두려워 하고 있다. 폭포모형의 앞단으로서 신속원형작성을 기업체들에 도입하는것은 관리자측에 련관된 위험요소들을 최소화하면서 그 방법에 접근할 기회를 제공해 준다.

신속원형작성모형은 10장에서 더 자세히 분석된다. 10장에서는 요구사항확정단계가 서술된다. 다른 부류의 생명주기모형을 이제 설명한다.

3．4．증식모형

쏘프트웨어는 작성되는것이 아니라 구성된다. 즉 쏘프트웨어는 건물이 구성되는것과 같은 방식으로 한계단씩 구성된다. 쏘프트웨어제품이 개발과정에 있는 동안에 매 계단들은 그전에 진행한 단계에 추가된다. 하루는 설계를 확장하고 다음날에는 또 다른 모듈을

코드작성한다. 완성된 제품의 구성은 완성될 때까지 이런 방식으로 점차적으로 진행된다.

물론 이러한 전진이 매일 진행되는것은 아니다. 어떤 계약자들이 때로 잘못 세워 진 벽들을 뜯어 버리거나 장식가들이 부주의로 깨뜨린 창유리를 교체해야 하는것과 마찬가지로 때때로 제품을 다시 명세작성하고 다시 설계하며 다시 코드를 작성해야 하는것, 최악의 경우에는 이미 완성된것을 버리고 다시 시작하여야 한다. 그러나 제품이 때로는 조화롭게 전진한다는 사실은 쏘프트웨어제품이 한 부분씩 구성된다고 하는 기본적인 현실을 부정하지는 않는다.

쏘프트웨어가 증식적으로 개발된다는 현실은 쏘프트웨어개발의 이러한 측면을 리용하는 모형 즉 그림 3-4에서 보여 준 이른바 증식모형(*incremental model*)의 개발을 초래하였다.

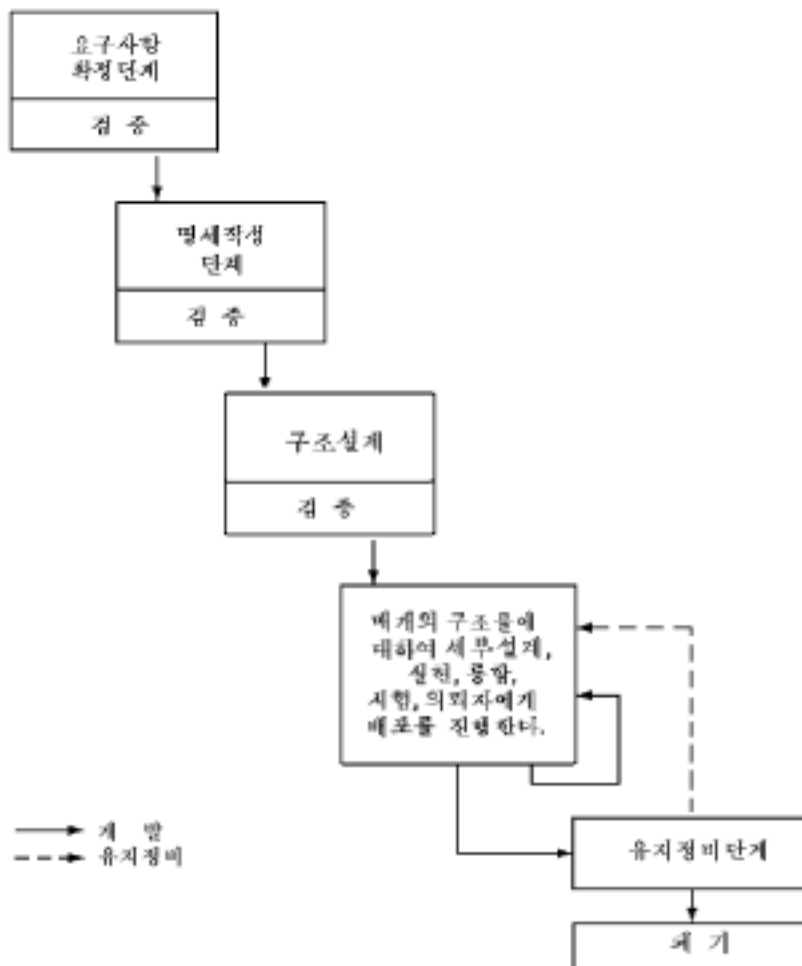



그림 3-4. 증식모형

제품은 일련의 점증적인 구조물(*build*)로서 설계, 실현, 통합되여 시험되는데 여기서

구조물은 특정한 기능을 수행할수 있도록 호상작용하는 여러가지 모듈들로부터 작성한 코드부분들로 이루어 진다. 실례로 만일 제품이 핵잠수함을 조종하는것이라면 그때 항해체계는 무기조종체계로 될수 있는 구조물로 구성한다. 조작체계에서는 일정작성기가 하나의 구조물로 될수 있으며 따라서 파일관리체계로 될수 있게 된다. 증식모형의 매 단계에서 새로운 구조물은 코드작성되고 그다음 자체로 시험되는 구조물로 통합된다. 이러한 과정은 제품이 목적하는 기능을 달성했을 때 즉 제품이 해당한 명세서를 만족시킬 때 중지된다. 개발자들은 적당하다고 생각할 때 목표제품을 구조물들로 자유롭게 분할한다. 이때 유일한 제약조건은 매 구조물이 현존 쏘프트웨어로 통합될 때 결과적인 제품이 시험가능하여야 한다는것이다. 만일 제품이 너무 적은 구조물들로 분할되면 증식모형은 구성 및 수정방법으로 퇴화된다(3.1). 반대로 만일 제품이 너무 많은 구조물들로 구성되면 매 단계에서 적은 량의 추가적인 기능을 통합시험하는데만도 상당한 시간이 소비된다. 제품을 구조물들로 최량분해하는것은 제품마다 개발자마다 다르다. 증식모형의 우점과 약점은 이제 서술하기로 한다.

3. 4. 1. 증식모형의 분석

폭포모형과 신속원형작성모형의 목적은 모두 완전하고 조작성이 좋은 제품을 의뢰자에게 배포하는것이다. 즉 의뢰자는 모든 요구를 만족시키는 제품을 가지게 되며 그에 대한 리용준비를 한다. 폭포모형이나 신속원형작성모형을 정확히 리용하는것은 철저히 시험되고 의뢰자가 제품이 원래의 개발목적으로 리용될수 있다고 확신하게 되는 그러한 제품을 생성하게 한다. 더우기 제품은 적당한 문서와 함께 제공되는데 이러한 문서들은 의뢰자측에서 리용될수 있을뿐아니라 필요에 따라서 세가지 류형의 유지정비 즉 적응, 완전 및 정정(1.3)유지정비가 진행될수 있게 한다. 두 모형에는 다 계획된 배포날자가 있다. 주의를 돌려야 할것은 제 날자에 또는 그 날자전에 주문작업을 완전히 완성하여 완성된 제품을 배포하는것이다.

이와 대비적으로 증식모형은 매 단계에서 오직 의뢰자의 요구를 부분적으로만 만족시키는 조작성이 좋은 제품을 배포하게 한다. 완성된 제품은 구조물들로 분할되며 개발자들은 제품을 구조물별로 배포한다. 전형적인 제품은 보통 5~25개의 구조물들로 구성된다. 매 단계에서 의뢰자는 요구의 필요한 몫을 수행하는 조작성이 좋은 제품을 가지게 된다. 즉 첫 구조물의 배포로부터 의뢰자는 유용한 작업을 할수 있게 된다. 증식모형을 리용하면 전체 제품의 구성부분들은 몇 주내에 쓸수 있게 되며 반면에 폭포모형이나 신속원형작성모형을 리용하여 개발한 제품을 받기 위하여 의뢰자들은 일반적으로 여러 달 또는 여러 해동안 기다리게 된다.

증식모형의 또 하나의 우점은 의뢰자측들에 완전히 새로운 제품을 강요하는 외부적인 영향을 줄이는것이다. 증식모형을 통한 제품의 점차적인 도입은 의뢰자들에게 새로운 제품을 조절할 시간을 제공한다. 변경은 매개의 확대되는 기업체들에 필수적인 부분으로 된다. 왜냐하면 쏘프트웨어제품은 현실의 모형이기때문에 변경에 대한 요구는 배포된 쏘프트웨어에 대하여 필수적인 부분으로 된다. 변경과 적응은 증식모형에 있어서 본질적인것인데 반면에 변경은 제품이 개발되여 하나의 큰 단계에 도입될 때 우환거리

로 될수 있다.

의뢰자들의 재정적견지에서 보면 단계적인 배포는 큰 자금지출을 요구하지 않는다. 대신 만일 최초의 구조물이 투자에 대한 높은 리윤을 준다는데 기초하여 선택된다고 하면 그이상의 현금류출이 있게 된다. 증식모형의 우점은 투자에 대한 리윤을 얻기 위하여 제품을 완성할 필요가 없다는것이다. 대신에 의뢰자는 임의의 시각에 제품개발을 중지할 수 있다.

증식모형의 한가지 난점은 어쨌든 매개의 추가적인 구조물이 그날까지 개발된것을 파괴하지 않고 현존구조에 병합되여야 한다는것이다. 더우기 현존구조는 이런 방법으로 확장하는데 적합하여야 하며 매개의 련속적인 구조물들에 대한 추가는 간단하고 단순해야 한다. 비록 이와 같은 열린 구성방식에 대한 요구는 확실히 단기적인 난점이여도 장기적으로 보면 하나의 실제적인 우점으로 될수 있다. 매 제품은 유지정비와 함께 개발된다. 개발기간에 명백한 명세서와 론리정연하고 응집도가 강한 설계를 작성하는것이 실로 중요하다. 그러나 일단 제품이 유지정비단계에 들어 가면 제품에 대한 요구들은 변화되며 본래의 확장은 이후의 유지정비가 불가능하게 될 정도로 론리정연하고 응집된 설계를 쉽게 파괴해 버릴수 있다. 이런 경우에 제품은 사실 령상태에서부터 다시 개발되여야 한다. 설계가 열려 져야 한다는것은 개발단계에서 선택된 모형에 관계없이 증식모형을 리용하는 제품개발에서 필수적인것이 아니라 유지정비에서 필수적인것이다. 이리하여 비록 증식모형이 폭포모형과 신속원형작성모형보다 더 주의 깊은 설계를 요구할수 있다고 하더라도 유지정비단계에서 보상된다. 만일 설계가 증식모형을 지원하는데 충분히 적응성이 있다면 설계는 제품을 붕괴시키지 않고 임의의 종류의 유지정비를 할수 있게 된다. 사실상 증식모형은 제품을 개발하는것과 그것을 확장(유지정비)하는것을 구별하지 않는다. 즉 매 확장은 단지 추가적인 구조물로 될뿐이다. 개발이 진행되고 있는 동안은 요구사항이 자주 변경된다. 즉 이 문제에 대해서는 16.4.4에서 더 자세히 론의한다. 증식모형의 적응성은 이와 관련하여 폭포모형과 신속원형작성모형을 초월한 뚜렷한 우점을 가져다 준다. 부정적인 측면으로서 증식모형은 아주 쉽게 구성 및 수정모형으로 퇴화되여 버릴수 있다. 공정전반에 대한 조종이 류실되며 열린 체계로 되지 못한 결과적인 제품은 유지정비자들에게는 암담한것으로 된다. 어떤 점에서는 증식모형이 용어상 모순적이다. 즉 앞으로의 확장을 포함하여 전체적인 제품을 지원할 어떤 설계를 시작하기 위하여 그 제품을 전체적으로 볼것을 개발자에게 요구하는것과 동시에 제품을 서로 독립인 구조물들의 렬로서 볼것을 요구한다. 만일 이런 명백한 모순을 조종하는데 충분히 숙련되지 않으면 증식모형은 불만족한 제품을 초래할수 있다.

그림 3-4에 있는 증식모형에서는 요구사항확정과 명세작성, 구성방식설계모두가 여러가지 구조물의 실현을 시작하기전에 완성되여야 한다. 보다 위험한 증식모형판본을 그림 3-5에 보여 주었다. 일단 의뢰자의 요구가 로출되면 첫 구조물에 대한 명세가 작성된다. 이것이 완성되면 명세작성팀은 설계팀이 그 첫번째 구조물을 설계하는 동안 두번째 구조물에 대한 명세작성에로 넘어 간다.

이리하여 매 팀이 이전의 구조들에서 얻는 정보들을 리용하면서 여러가지 구조물들을 병렬로 개발한다. 이러한 방법은 결과적인 구조물들이 서로 어울리지 않을것이라는 실제적인 위험을 초래한다. 그림 3-4의 증식모형에서는 명세 및 구성방식설계가 첫 구조

물이 시작되기전에 완성되여야 한다는 요구는 처음부터 전반적인 설계를 진행하여야 한다는것을 의미한다. 그림 3-5의 병행증식모형에서 개발공정이 주의 깊게 검사되지 않으면 전체적인 프로젝트의 위험요소들이 붕괴된다. 그럼에도 불구하고 이러한 병행모형은 일부경우에 성공하였다. 실례로 큰 규모의 통신체계를 개발하기 위하여 후지쯔회사에서 그것을 리용하였다[Aoyama, 1993].

3. 5. 최종적프로그람작성

최종적인 프로그람작성은[Beck, 1999] 증식모형에 기초하고 있는 쏘프트웨어생산에 대한 론의할만한 새로운 방법이다. 첫번째 단계는 의뢰자들이 제품에 지원하기를 바라는 여러가지 특성들을 쏘프트웨어생산팀이 결정하는것이다. 이런 매 특성에 대하여 팀에서는 의뢰자에게 그런 특성을 실현하는데 시간은 얼마나 걸리고 비용은 얼마인가를 알려 준다. 이 첫번째 단계는 증식모형에서 요구사항확정과 명세작성단계에 해당된다(그림 3-4).

의뢰자는 비용 대 리득분석(5.2)을 리용하여 즉 자기의 업무특성에 대한 잠재적인 리득은 물론 개발팀이 제공하는 시간과 비용에 대한 타산에 토대하여 매개의 련이은 구조물에 포함되여야 할 특성들을 선택한다. 제안된 구조물은 보다 더 작은 부분 즉 과제(*task*)들로 쪼개여 진다.

프로그람작성자는 우선 과제에 대한 시험실례를 작성한다. 그다음 한 화면상에서 어떤 동업자들과 함께 작업하여(2인프로그람작성; *pair programming*)[Williams, Kessler, Cuningham, and Jeffries, 2000] 모든 시험실례가 정확히 동작한다는것을 담보하면서 과제를 실현해 나간다. 과제는 그 다음제품의 현재판본안에 통합된다. 리상적으로 과제를 실현하고 통합하는데는 몇시간이상 걸리지 않는다. 보통 여러 2인조들이 과제들을 병렬로 실현하며 통합은 련속적으로 진행된다. 과제에 대하여 리용되는 시험실례들은 이후의 모든 통합시험에서 유지되고 리용된다. 최종적프로그람작성(XP)의 여러 특성들은 약간 독특하다. 즉

1. XP개발팀의 콤퓨터들은 작은 방들로 칸을 막은 큰 방에 설치된다.
2. 의뢰자의 대표자는 전 기간 XP개발팀과 함께 일을 한다.
3. 개별적인 사람들은 두주동안 련속적으로 일할수 없다.
4. 전문화는 없으며 대신 XP개발팀의 모든 성원들은 명세작성과 설계, 코드작성 그리고 시험에 종사한다.
5. 그림 3-5에 보여 준 보다 위험한 증식모형에서와 같이 여러가지 구조물들이 개발되기전에 전면적인 설계단계는 존재하지 않는다. 대신 설계는 제품이 개발되는 동안 수정된다. 이러한 처리절차를 재인자분해(*refactoring*)라고 한다. 시험실례가 동작하지 않을 때마다 실례가 단순하고 명백하며 모든 시험실례들이 만족하게 동작한다고 개발팀이 인정할 때까지 코드는 재조직된다.

XP는 많은 소규모 또는 중규모프로젝트들에서 성공적으로 리용된다. 이러한 프로젝트들에는 9명이 한달동안 배수송에서의 관세계산체계를 개발하는것으로부터 100명이 1년동안 가격분석체계를 개발하는것들이 포함된다[Beck, 1999]. XP의 우점은 의뢰자들의 요구가 명확치 않거나 변화될 때 리용될수 있다는것이다. 그러나 XP는 아직은 이러한 판본

의 증식모형이 그것의 초기계약을 완성할것인가를 결정할수 있을 정도로 광범히 리용되지 못하고 있다.

3. 6. 동기 및 안정화모형

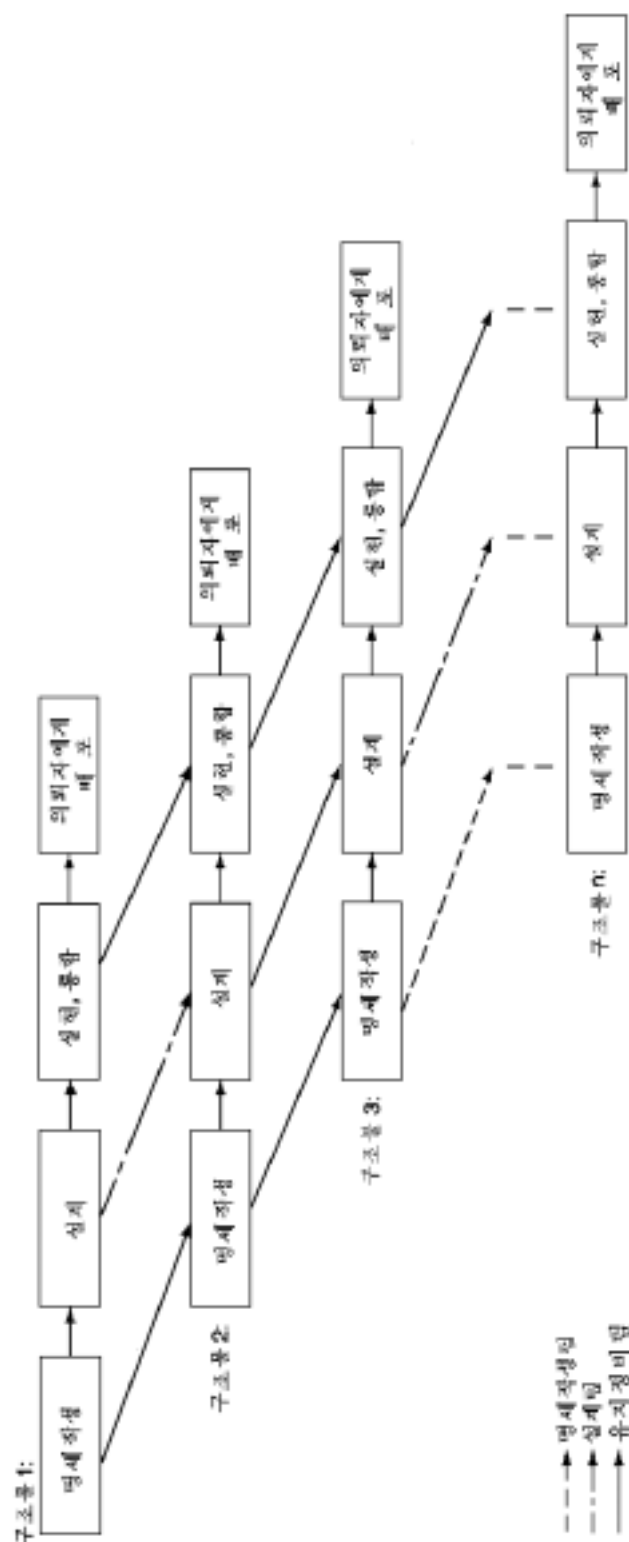
마이크로쏘프트회사는 상업적인 규격쏘프트웨어를 제작하는 세계최대의 회사이다. 그 패키지의 대부분은 동기 및 안정화모형(*synchronize-and-stabilize*)이라고 하는 증식모형의 한가지 판본을 리용하여 개발된다[Cusumano and Selby, 1997].

요구사항분석단계는 이 패키지의 많은 잠정적사용자들과 면담을 하고 사용자들이 설정하는 우선권을 가진 특성목록을 추출하는것에 의해서 관리된다. 명세서가 작성된다. 다음으로 작업은 셋 또는 네개의 구조물들로 분할된다. 첫번째 구조물은 가장 중요한 특성들로 이루어 지며 두번째 구조물은 그다음 중요한 특성들로 이루어 지는 방식으로 계속 진행된다. 매 구조물들은 병렬로 작업하는 여러개의 작은 개발팀들에 의하여 실현된다. 하루사업의 마감에 모든 팀들은 동기화된다. 즉 그들은 부분적으로 완성된 구성요소들을 한데 모아 놓고 결과적인 제품을 시험하고 수정한다. 안정화(*Stabilization*)는 매 구조물들의 마감에 진행된다. 발견된 나머지 모든 오유들은 수정되고 구조물은 고정된다. 즉 명세서는 더이상 변경되지 않는다. 반복되는 동기화단계는 여러개의 구성요소들이 함께 동작한다는것을 담보한다. 부분적으로 구축된 제품에 대하여 이러한 규칙적인 실행을 진행하는 또 하나의 우월성은 개발자들이 제품의 조작에 대하여 신속하게 간파할수 있고 필요한다면 구축이 진행되는 과정에 요구사항을 수정할수 있다는것이다. 만일 초기의 명세서가 불완전하다면 이 모형도 리용될수 있다. 동기 및 안정화모형은 4.5에서 더 고찰되는데 여기서는 팀의 세부에 대하여 론의한다.

라선모형은 그것이 모든 다른 모형들의 여러 측면을 병합한것이기때문에 마지막에 취급한다.

3. 7. 라선모형

쏘프트웨어개발과정에는 거의 언제나 항상 위험한 요소들이 포함된다. 실례로 책임직원은 제품이 적당하게 문서화되기전에 다시 수표를 할수 있다. 제품이 결정적으로 의존하고 있는 하드웨어제작자는 파산에 직면할수 있다. 시험과 품질보증에 너무 많은 자금이 또는 너무 적은 자금이 투자될수 있다. 주요 쏘프트웨어제품을 개발하는데 수십만딸라를 소비한 다음 기술적인 돌파구들이 전체적인 제품을 쓸모 없게 만들수 있다. 어떤 개발기업체는 자료기지관리체계을 연구개발할수도 있다. 그러나 제품이 시장에 나가기전에 경쟁자들이 보다 낮은 가격을 가진 기능적으로 등가인 패키지를 발표하게 된다. 그림 3-5의 증식모형을 리용하여 구축된 제품의 구성부분들은 서로 어울리지 않을수 있다. 쏘프트웨어개발자들은 명백한 리유로 하여 이러한 위험성을 가능한껏 최소화하려고 한다.

그림 3-5. 보다 더 위험이 병행되는 점증식모형

이러한 확실한 위험성들을 최소화하는 한가지 방도는 원형을 작성하는것이다. 3.3에서 서술한바와 같이 배포된 제품이 의뢰자들의 실제적인 요구를 만족시키지 않는다는 위험요소들을 감소시키는 좋은 방도는 요구사항확정단계에서 신속원형을 작성하는것이다. 그 다음단계들에서 다른 종류의 원형들이 만들어 진다. 실례로 전화회사는 장거리망을 통하여 호출을 진행할수 있는 새로운 효률 높은 알고리듬을 창안할수 있다. 만일 제품이 실현되였으나 기대했던대로 동작을 하지 못하면 전화회사는 제품을 개발하는데 든 비용을 랑비하는것으로 된다. 이밖에 성이 나거나 또는 못마땅해 하는 사용자들은 자기의 업무를 다른 곳에 맡길수도 있다. 이러한 대본은 호출의 경로결정만을 조종하는 하나의 원형을 작성하고 그것을 어떤 모의기에서 시험하는 방식으로 피할수 있다. 이런 방식으로 실제체계는 혼란되지 않으며 경로결정알고리듬을 실현하는 비용에 관하여 전화회사는 이 새로운 알고리듬을 병합하고 있는 어떤 전체적인 망조종기를 개발할 가치가 있는가를 결정할수 있다.

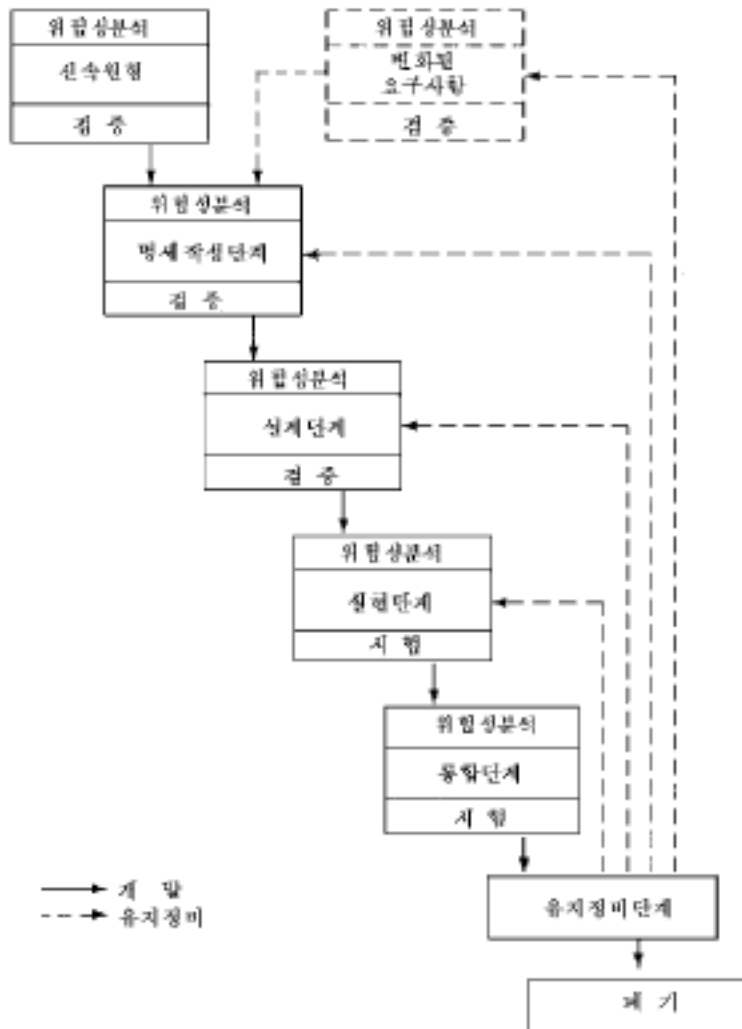



그림 3-6. 간단한 라선모형

원형들과 기타 수단들을 리용하여 위험성요소들을 최소화하는 착상은 라선(*spiral*)모형의 기초를 이루고 있는 개념이다[Boehm, 1988]. 이러한 생명주기모형을 고찰하는 한가지 간단한 방법은 그림 3-6에서 보여 준바와 같이 매 단계에서 하나의 폭포모형을 위험성분석에 선행시키는 방법이다(그림의 일부분은 라선모형을 반영하도록 이 그림에 해당한 부분이 그림 3－7로 다시 그려 졌다.).

그림 3-7. 라선형으로 다시 그린 그림 3-6의 부분

매 단계를 시작하기전에 위험성을 조종(또는 해소)하기 위한 시도가 진행된다. 이 단계에서 중요한 모든 위험성을 해소하는것이 불가능하면 프로젝트는 즉시에 끝난다.

원형은 어떤 부류의 위험에 대한 정보들을 제공하는데 효과적으로 리용될수 있다. 실례로 시간적인 제약은 보통 원형을 작성하고 그 원형이 필요한 성능을 실현할수 있는가를 측정함으로써 시험할수 있다. 만일 원형이 제품의 관련된 특징들에 대한 하나의 정확한 기능적인 표현으로 된다면 원형에 대하여 진행되는 측정은 개발자들에게 시간적인 제약의 달성여부에 관한 하나의 훌륭한 착상을 주게 된다.

다른 령역에서의 위험요소들은 원형작성을 보다 어렵게 한다. 실례로 제품을 개발하는데 필요한 쏘프트웨어직원들이 고용될수 없다거나 또는 책임직원이 제품이 완성되기전에 다시 수표하는 그러한 위험요소들이 자주 발생한다. 또 하나의 잠재적인 위험은 어떤 특정한 개발팀이 하나의 특정한 대규모제품을 개발할수 있을 정도의 자격이 없다는것이다. 어떤 가정에서 리용하는 제품들을 개발하는데서 성공한 계약자들은 큰 사무소에서 리용하는 복잡한 제품들을 개발할수 없을수도 있다. 마찬가지로 소규모

쏘프트웨어와 대규모쏘프트웨어사이에는 본질적인 차이가 있으며 원형작성은 거의 리용되지 않는다. 이러한 위험요소들은 어떤 훨씬 작은 파라다임에 대하여 팀의 성능을 시험함으로써 해결될수 없으며 이 파라다임에서 대규모쏘프트웨어에 특정한 팀조직에 대한 문제점들은 발생할수 없다. 원형작성이 진행될수 없는 또 한가지 위험은 하드웨어공급자들의 배포계약을 타산하는것이다. 개발자들이 채용할수 있는 하나의 전략은 공급자들의 이전의뢰자들이 얼마나 잘 취급되는가를 결정하는것이지만 과거의 성능이 결코 앞으로의 성능을 명백하게 예언하여 주는것은 아니다. 배포계약서에서 벌금에 대한 조항은 필수적인 하드웨어가 제정된 시간에 배포될것이라고 담보하는 한가지 방도로 되지만 만일 공급자들이 그러한 조항을 포함시키는데 동의하지 않는다면 어떻게 되겠는가? 지어 벌금조항에 따라서 지연된 배포가 발생할수도 있으며 결국에는 여러해동안 법적소송을 당할수 있는 재판에 기소될수도 있다. 그럭저럭하는 사이에 계약된 하드웨어의 미배포가 계약된 쏘프트웨어의 미배포를 발생시키는것으로 하여 쏘프트웨어개발자들은 파산에 직면하게 된다. 간단히 말하면 원형작성이 일부 같은 령역의 위험요소들을 감소시키는 반면에 다른 령역의 위험요소들은 기껏해서 부분적으로 해소할수 있으며 또 다른 령역의 위험요소들은 전혀 해소할수 없을수도 있다. 라선모형의 총체적인 륜곽을 그림 3-8에 보여 준다.

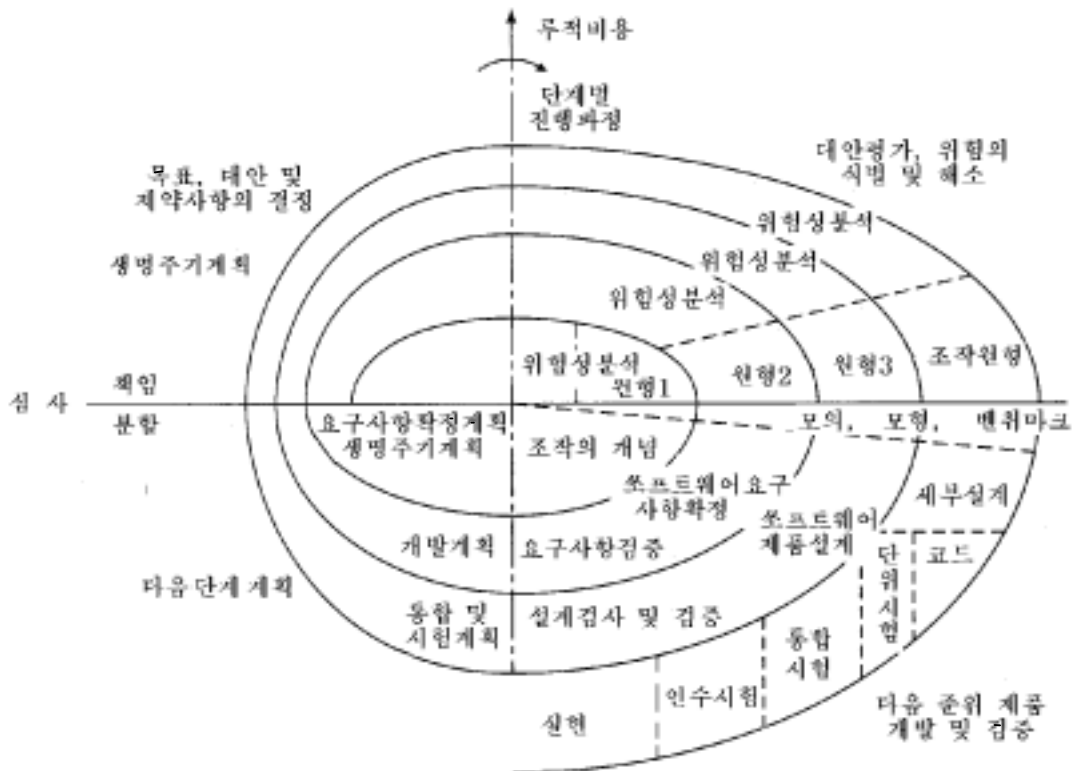



그림 3-8. 완전한 라선모형 [Boehm,1988].(©1988, IEEE.)

반경은 날자별 루적비용을 나타내며 각도는 라선모양의 전진과정을 나타낸다. 매 라선의 주기는 단계에 대응한다. 매 단계는 그 단계의 목적과 그 목적을 달성하기 위한 방도 그리고 방도들에 대한 제약들을 결정하는것으로써 시작된다(상단 왼쪽 4.4분구). 이 공정은 이 목적들을 달성하기 위한 하나의 전략을 초래한다. 다음으로 그 전략은 위험성의 견지에서 분석된다. 일부 경우에 하나의 원형을 구성함으로써 모든 잠재적인 위험요소들을 해석하기 위한 시도들이 진행된다. 만일 위험요소가 명백히 해소될수 없다고 하면 프로젝트는 그 즉시에 끝나고 만다. 그러나 일부 정황하에서 이 프로젝트를 어떤 훨씬 작은 규모로 계속하여 진행하기 위한 결심이 이루어 질수 있다. 만일 모든 위험요소들이 성공적으로 해소되면 그 다음개발단계가 시작된다(하단 오른쪽 2.4분구). 라선모형의 이 4개분구는 순수한 폭포모형에 대응한다. 마지막으로 그 단계의 결과들이 평가되고 다음단계가 계획된다.

라선모형은 넓은 분야의 제품들을 개발하는데 성과적으로 리용되였다. 라선모형이 생산성을 높이기 위한 다른 방법들과 결합되여 리용된 25개의 프로젝트에서 매 프로젝트의 생산성은 이전보다 적어도 50%까지 늘어 났으며 대부분의 프로젝트들에서는 100%까지 늘어 났다[Boehm, 1988]. 라선모형이 주어 진 프로젝트에 대하여 리용되여야 하는가를 결정하기 위하여 그것의 우점과 약점에 대하여 평가한다.

3. 7. 1. 라선모형의 분석

라선모형은 여러가지 우점을 가지고 있다. 다른 대안들과 제약에 대한 강조는 현존 쏘프트웨어의 재리용(8.1)과 특정한 목적으로서의 쏘프트웨어품질의 병합을 지원한다. 이밖에 쏘프트웨어개발에서의 일반적인 문제들은 어떤 특정한 단계에서의 제품이 적당하게 시험될 때에 결정된다. 시험에 많은 시간을 소비하는것은 곧 돈을 랑비하는것이며 제품의 배포가 지나치게 지연되게 할수도 있다. 반대로 시험이 진행되면 배포된 쏘프트웨어는 잔류오유를 포함하고 있으므로 개발자들에게 기분 나쁜 결과를 가져다 준다. 라선모형에서는 충분한 시험을 진행하지 않거나 또는 지나치게 시험을 진행함으로써 위험요소들이 발생한다. 아마도 라선모형의 구조내에서 가장 중요한 유지정비는 단순히 또 하나의 라선형주기로 된다. 즉 본질에 있어서 유지정비와 개발사이에는 구별이 없다. 이리하여 무식한 쏘프트웨어전문가들에 의하여 때때로 유지정비가 손해를 보는것과 같은 문제는 제기되지 않는다. 왜냐하면 유지정비는 개발과 같은 방법으로 취급되기때문이다.

라선모형의 응용에 대한 제약이 존재한다. 특히 그 표현형식에서 모형은 대규모쏘프트웨어의 내부적개발에 있어서 배제되는 경향이 있다[Boehm, 1988]. 내부적인 프로젝트 즉 개발자와 의뢰자가 모두 같은 기관의 성원으로 되는 경우를 고찰하자. 만일 위험분석결과 이 프로젝트가 종결되여야 한다는 결론에 도달하게 되면 기관내 쏘프트웨어개발자들은 어떤 다른 프로젝트를 수행하도록 배치될수도 있다. 그러나 일단 개발기업체와 외부의뢰자사이에 계약이 체결되면 어느 한 측이 이 계약을 끝내려고 하는 시도는 계약위반에 대한 법적소송을 초래할수 있다. 그렇기때문에 계약쏘프트웨어인 경우에 모든 위험분석은 계약이 체결되기전에 의뢰자와 개발자들에 의하여 수행되여야 하며 라선모형에서처럼 진행되지 말아야 한다.

라선모형에 대한 두번째 제약은 프로젝트의 크기와 관련되여 있다. 특히 라선모형은 대규모쏘프트웨어에만 적용될수 있다. 만일 위험분석을 진행하는 비용이 프로젝트전체의 비용과 맞먹거나 또는 위험분석을 진행하는것이 잠재적인 리득에 크게 영향을 주게 된다면 위험분석을 진행하는것은 아무런 의미도 없다. 그대신 개발자들은 위험요소들에 얼마나 많은 비용이 소비되는가를 결정하고 그다음 위험분석을 진행하는데 얼마만한 비용이 소비되는가를 결정하여야 한다.

라선모형의 한가지 중요한 우점은 그것이 위험요소에 의하여 구동된다는것인데 이것은 또한 하나의 약점으로 될수도 있다. 만일 쏘프트웨어개발자들이 가능한 위험요소들을 지적하고 그 위험요소들을 정확하게 분석하는데 숙련되여 있지 않다면 프로젝트가 사실상 불행에 빠지게 될 때에 개발팀이 모든것은 잘 진행되고 있다고 믿게 되는 실제적인 위험성이 존재한다. 만일 개발팀성원들이 충분한 자격이 있는 위험요소분석가들이라면 관리자측은 라선모형을 리용하기로 결정하게 된다.

3. 8. 객체지향생명주기모형

객체지향파라다임을 리용한 경험은 공정의 단계들 또는 단계의 부분들사이에서 진행되는 반복이 구조화파라다임을 리용할 때보다 객체지향파라다임을 리용할 때 보다 더 일반적인 현상으로 된다는것을 보여 주었다. 객체지향생명주기모형은 명백하게 반복에 대한 요구를 반영하도록 제한되였다. 이와 같은 하나의 모형은 그림 3-9에서 보여 준 분수모형이다[Henderson-Sellers and Edwards, 1990].

여러가지 단계를 나타내는 원들은 활동들사이의 겹침을 명백히 반영하면서 겹쳐 있다. 단계안의 화살표들은 그 단계안에서의 반복을 나타낸다. 유지정비에 관한 원은 객체지향파라다임이 리용될 때 유지정비에 대한 노력이 감소되는것을 나타내기 위하여 더 작게 그려 주었다.

분수모형외에 재귀/병렬생명주기[Berard, 1993]와 원형려행형태설계[Booch, 1994] 그리고 통합쏘프트웨어개발공정에 토대한 모형[Jacobson, Booch, and Rumbaugh, 1999]을 비롯하여 여러가지 객체지향생명주기모형들이 제안되였다. 이 모든 모형들은 반복적이며 병렬적인 일련의 형태들을 병합하고 있으며 증식개발(3.4)을 지지하고 있다. 이러한 생명주기모형의 위험은 다음과 같다. 그것들이 필요이상의 효과를 얻기 위한 시도로서 오해될수 있으며 또 그로 말미암아 개발팀성원들이 처음에는 제품의 어느 한 부분을 설계하고 다음에는 또 다른 부분을 분석하며 그다음 전혀 분석되지도 설계되지도 않은 세번째 부분을 실현하는 방식으로 매개 단계들사이를 우연적으로 이동하는 완전히 비학문학적인 쏘프트웨어개발형식에로 이끌어 갈수 있다는것이다. 다음의 《알고 싶은 문제》에서는 이와 같은 희망하지 않는 방법에 대하여 보다 상세히 보여 주고 있다. 더 좋은 방법은 객체지향파라다임의 실체는 바로 주기적인 반복과 세련이 명백히 요구된다는것을 옳게 인식하는것외에 선형공정(3.3에서의 신속원형작성모형이나 또는 그림 3-9에서의 중심수직선과 같은)을 하나의 전체적인 목적으로 취하는것이다. 이 문제는 단순히 객체지향파라다임과 련관된 한가지 새로운것이라는것이 제기될수도 있다.

알고 싶은 문제

개발팀성원들이 하나의 과제로부터 다른 과제로 아무렇게나 이동한다면 이것은 때때로 CABTAB(하나의 비트를 코드작성하고 하나의 비트를 시험한다.)로서 간주된다. 초기에는 이 략어가 객체지향파라다임과 결합하여 리용된것과 같은 성공적인 반복모형과 관련하여 긍정적인 의미로서 리용되였다. 그러나 단어 해커(*hacker*)가 원래의 의미외에 하나의 멸시적인 의미를 가지는것과 마찬가지로 CABTAB도 지금 쏘프트웨어개발에 대한 비학문적인 방법들과 관련한 경멸적인 의미로 리용되고 있다.

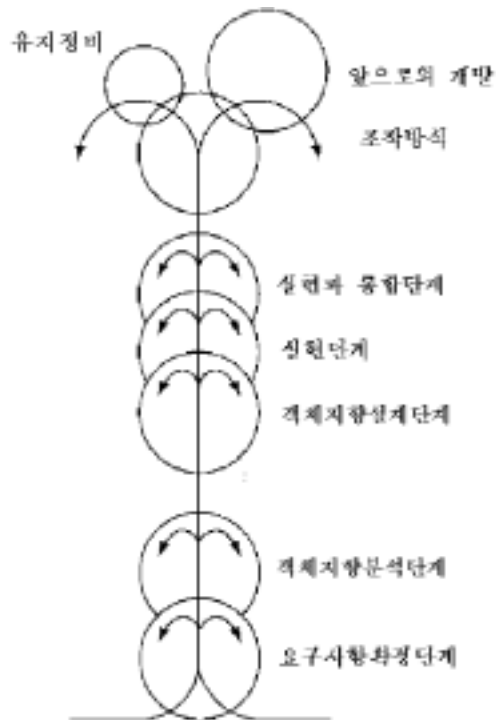



그림 3-9. 분수모형

쏘프트웨어전문가들이 객체지향분석과 객체지향설계에 대하여 보다 더 경험을 가지게 될때 론리는 타당하며 전반적학문이 성숙될 때 반복적인 검토와 수정에 대한 요구는 줄어들게 된다. 이러한 론증이 불합리하다는것을 보기 위하여 이 장에서 이미 서술한 여러가지 개발모형들을 고찰하기로 하자. 우선 명백한 반결합고리를 가진 폭포모형(3.2)을 생각해 보자. 다음으로 주요한 개발은 신속원형작성모형(3.3)인데 그 주요한 목적의 하나는 반복에 대한 요구를 감소시키는것이였다. 그다음에 라선모형(3.7)이 나왔는데 이것은 명백히 쏘프트웨어개발과 유지정비에 대한 반복을 반영하고 있다. 이밖에 되돌이는 객체지향분석을 위한 코드-요르돈기법[Coad and Yourdon, 1991a]의 고유한 측면이라는것은 문헌 [Honiden, Kotaka and Kishimoto, 1993]에서 제시되였으며 보다 새로운 객체지향분석

기법에 대하여 역시 류사한 결과들이 성립하는것 같다. 달리 말하면 반복은 일반적으로는 쏘프트웨어생산의 고유한 속성이며 특히 객체지향파라다임에서 그러하다.

3. 9. 생명주기모형의 비교

6개의 서로 다른 클라스에 대한 쏘프트웨어생명주기모형들을 그것들의 우단점에 특별한 주목을 돌리면서 시험하였다. 구성 및 수정모형(3.1)은 피하여야 한다. 폭포모형(3.2)은 이미 알려 진 개념이다. 그것의 우점은 리해되며 그래서 그것의 약점도 리해된다. 신속원형작성모형(3.3)은 배포제품이 실지 의뢰자가 요구한것이 아닐수 있다고 하는 폭포모형에서의 특정한 약점에 대처하여 개발되였다. 폭포모형은 잘 알려 져 있지만 새롭게 나온 신속원형작성모형에 대해서는 잘 알려 져 있지 않다. 신속원형작성모형은 제10장에서 보여 준바와 같이 일정한 자체의 문제점을 가질수 있다. 하나의 다른 방도는 3.3.1에서 제기한바와 같이 두 모형의 우점을 결합하는것이다. 이 모형은 성공적임에도 불구하고 일부 결함을 가지고 있다. 최종적인 프로그람작성(3.5)은 론의할만한 방법이다. 동기 및 안정화모형(3.6)은 마이크로쏘프트회사에서 큰 성과를 보았다. 그러나 다른

생명주기모형	우 점	약 점
구성 및 수정모형 (3.1)	유지정비를 요구하지 않는 작은 프로그람에서는 좋다.	중요한 프로그람에 대하여서는 전혀 충분하지 않다.
폭포모형 (3.2)	문서구동숙련방법	배포된 제품은 의뢰자의 요구를 만족시키지 않는다.
신속원형작성모형 (3.3)	배포된 제품은 의뢰자의 요구를 만족시킨다.	모든 의심이 증명되지 않았다.
증식모형 (3.4)	초기투자가 최대로 된다. 유지정비가능성을 촉진시킨다.	열린 구성방식을 요구한다. 구성 및 수정모형으로 퇴화될수 있다.
최종프로그람작성 (3.5)	초기 투자가 최대로 된다. 의뢰자의 요구가 모호할 때 잘 작업한다.	아직 널리 리용되지 않는다.
동기 및 안정화모형 (3.6)	미래의 사용자의 요구를 만족시킨다. 구성요소들을 성과적으로 통합할수 있다.	다른 기업체들에서는 마이크로쏘프트회사보다 널리 리용하지 않고 있다.
라선모형 (3.7)	우의 모든 모형의 특징을 병합한다.	큰 규모의 내부리용제품들에만 리용가능하다. 개발자들은 위험성분석 및 위험해소능력이 있어야 한다.
객체지향모형 (3.8)	단계안에서, 단계들사이에 반복가능하다.	CABTAB로 퇴화될수 있다.

그림 3-10. 3장에서 설명한 생명주기모형의 비교와 취급한 절

기업분야에서는 뚜렷한 성과가 증명되지 않았다. 또 다른 방도는 라선모형(3.7)을 리용하

는것인데 그것은 개발자들이 위험분석과 위험해소에 충분히 숙련되였을 때에만 가능하다. 고려해야 할 요인은 객체지향파라다임이 리용될 때 생명주기모형이 반복적이여야 하며 즉 그것이 반결합을 지원해야 한다는것이다(3.8). 이 장에서 취급한 여러가지 생명주기모형의 우단점은 그림 3-10에 개괄되여 있다.

매 쏘프트웨어개발기업체는 그 조직과 경영, 종업원들과 쏘프트웨어공정에 알맞는 생명주기모형을 결정하여야 하며 현재 개발중에 있는 특정한 제품의 성질에 따라 모형을 바꾸어야 한다. 이러한 모형은 그것의 우점은 리용하고 약점은 최소화하는 방식으로 여러가지 생명주기모형의 적절한 측면들을 병합할것이다.

요 약

구성 및 수정모형(3.1), 폭포모형(3.2), 신속원형작성모형(3.3), 증식모형(3.4), 최종적프로그람작성(3.5), 동기 및 안정화모형(3.6), 라선모형(3.7), 객체지향생명주기모형(3.8)을 비롯한 여러가지 생명주기모형들이 서술되였다. 3.9에서는 이러한 생명주기모형들을 비교대조하고 특정한 프로젝트에 대한 생명주기모형의 선택에 관한 제안들이 제시되였다.

보 충

폭포모형은 처음으로 문헌 [Royce, 1970]에서 제시되였다. 폭포모형에 대한 분석을 문헌 [Royce, 1998]의 1장에서 주었다.

신속원형작성에 대한 소개는 문헌 [Connell and Shafer, 1989]와 [Gane, 1989]에 제시되였다. 콤퓨터지원원형작성의 역할은 문헌 [Luqi and Royce, 1992]에서 찾아 볼수 있다. 1995년 2월에 발표된 *IEEE Computer*에 신속원형작성에 대한 몇개의 기사를 주었다. 증식모형의 한가지 류형인 진화적배포모형에 대한 설명은 문헌 [Gilb, 1988]에서 찾아 볼수 있다. 병행증식모형은 문헌[Aoyama, 1993]에 서술되여 있다. 동기 및 안정화모형은 문헌 [Cusumano and Selby, 1997]에서는 개괄하여 주었으며 문헌 [Cusumano and Selby, 1995]에서 상세히 서술하였다. 동기 및 안정화모형은 문헌 [McConnell, 1996] 에서 파악할수 있다. 라선모형은 문헌 [Boehm, 1988]에서 서술되였고 TRW쏘프트웨어생산체계에 대한 응용은 문헌 [Boehm et al., 1984]에서 찾아 볼수 있다. 최종적프로그람작성은 문헌 [Beck, 1999]와 문헌 [Beck, 2000]에서 서술되였고 재인자분해는 문헌 [Fowler et al., 1999]의 주제이다.

위험성분석은 문헌 [Boehm, 1991; Jones, 1994c; Karolak, 1996 ; and Keil, Cule, Lyytinen and Schmidt, 1998]에 서술되여 있다. *IEEE Software* 1997년 5월/6월호에 위험성관리에 관한 10개의 기사가 있다.

객체지향생명주기모형들은 문헌 [Henderson-Sellers and Edwards, 1990; Rajlich, 1994; and Jacobson, Booch, and Rumbaugh, 1999]에 서술되여 있다. 많은 다른 생명주기모형들이 제기되고 있다. 실례로 인간적인자들을 강조한 하나의 생명주기모형은 문헌 [Mantei and Teorey, 1988]에 서술되였으며 문헌 [Rajlich and Bennett, 2000]에서는 유지정비지향생명

주기모형을 서술하고 있다. 쏘프트웨어공학연구소에서 제기한 생명주기모형은 문헌 [Landis et al., 1992]에 서술되여 있다. *IEEE Software* 2000년 7월/8월호에는 최종적프로그람작성의 한 구성부분인 2인프로그람작성에 대한 실험을 서술하고 있는 문헌 [Williams, Kessler, Cunningham, and Jeffries, 2000]을 비롯한 쏘프트웨어생명주기모형에 대한 여러개의 론문들이 있다. 국제쏘프트웨어개발공정토론회 회보는 생명주기모형에 관한 정보를 주는 유용한 문헌으로 되고 있다. [ISO/IEC 12207, 1995]는 쏘프트웨어생명주기공정에 대한 표준으로 되고 있다.

문 제

3.1. 당신이 3.748571의 거꿀수를 소수점아래 4자리까지 계산하는 제품을 개발해야 한다고 가정하자. 일단 제품이 실현되여 시험되면 그 제품은 버리게 된다. 당신은 어떤 생명주기모형을 리용하려고 하는가? 대답과 그 리유를 밝히시오.

3.2. 당신은 쏘프트웨어공학고문인데 단음식을 생산하여 여러 식당들에 판매하는 회사인 《사멸되여 가는 단음식;*Deplorably Decadent Desserts*》의 재정부지배인에게 호출되였다. 그 녀자는 자기들이 생산하여 여러 식당들에 배포할수 있도록 여러가지 원료들을 사들이고 단음식들이 생산되여 식당들에 배달될 때 그것들을 조사하는것으로부터 시작하는 회사의 제품을 검사하는 제품을 개발할것을 당신의 기업체에 부탁한다. 당신은 그 프로젝트에 대한 생명주기모형을 선택하는데서 어떤 기준을 리용하겠는가?

3.3. 문제 3.2의 쏘프트웨어를 개발하는데서 위험요소들의 명세를 작성하시오. 매 위험요소을 해소하기 위하여 어떻게 하려고 하는가?

3.4. 회사《사멸되여 가는 단음식》을 위한 창고조종제품의 개발이 아주 성공적이다. 결과 이 회사는 소매기업체들은 물론 식당들을 위하여 음식을 준비하고 판매하는 여러 회사들에 팔아 줄 COTS패키지로서 제품이 다시 작성될것을 요구한다. 그러므로 새로운 제품은 새로운 하드웨어와 조작체계에 이식가능하고 쉽게 적응할수 있어야 한다. 문제 3.2의 대답에 있는 프로젝트와 차이나는 이 프로젝트에 대한 생명주기모형을 선택하는데서 어떤 기준을 리용하는가?

3.5. 증식모형에 대하여 리성적으로 응용할수 있는 제품의 종류를 서술하시오.

3.6. 증식모형이 난관에 부닥치게 되는 상황의 류형들을 서술하시오.

3.7. 라선모형에 리성적으로 응용할수 있는 제품의 종류를 서술하시오.

3.8. 라선모형이 적합치 않은 상황의 류형을 서술하시오.

3.9. 폭포와 분수에서 공통적인것은 무엇인가? 폭포모형과 분수모형에서 공통적인것은 무엇이며 그것들은 어떻게 다른가?

3.10. (과정안상 목표) 부록 1에 서술된 브로드랜즈지역 아동병원을 위해서는 어떤 쏘프트웨어생명주기모형을 리용하겠는가. 대답과 함께 그 리유를 밝히시오.

3.11. (쏘프트웨어공학독본) 교원이 문헌 [Beck, 1999]의 복제본을 배포해 줄것이다. 최종적프로그람작성방법을 리용하는 회사에서 일하고 싶은가?

참 고 문 헌

[Aoyama, 1993] M. AOYAMA, "Concurrent-Development Process Model," *IEEE Computer* **10** (July 1993), pp. 46–55.

[Beck, 1999] K. BECK, "Embracing Change with Extreme Programming," *IEEE Computer* **32** (October 1999), pp. 70–77.

[Beck, 2000] K. BECK, *Extreme Programming Explained: Embrace Change,* Addison-Wesley Longman, Reading, MA, 2000.

[Berard, 1993] E. V. BERARD, *Essays on Object-Oriented Software Engineering,* Volume 1, Prentice Hall, Englewood Cliffs, NJ, 1993.

[Boehm, 1988] B. W. BOEHM, "A Spiral Model of Software Development and Enhancement," *IEEE Computer* **21** (May 1988), pp. 61–72.

[Boehm, 1991] B. W. BOEHM, "Software Risk Management: Principles and Practices," *IEEE Software* **8** (January 1991), pp. 32–41.

[Boehm et al., 1984] B. W. BOEHM, M. H. PENEDO, E. D. STUCKLE, R. D. WILLIAMS, AND A. B. PYSTER, "A Software Development Environment for Improving Productivity," *IEEE Computer* **17** (June 1984), pp. 30–44.

[Booch, 1994] G. BOOCH, *Object-Oriented Analysis and Design with Applications,* 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.

[Coad and Yourdon, 1991a] P. COAD AND E. YOURDON, *Object-Oriented Analysis,* 2nd ed., Yourdon Press, Englewood Cliffs, NJ, 1991.

[Connell and Shafer, 1989] J. L. CONNELL AND L. SHAFER, *Structured Rapid Prototyping: An Evolutionary Approach to Software Development,* Yourdon Press, Englewood Cliffs, NJ, 1989.

[Cusumano and Selby, 1995] M. A. CUSUMANO AND R. W. SELBY, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People,* The Free Press/Simon and Schuster, New York, 1995.

[Cusumano and Selby, 1997] M. A. CUSUMANO AND R. W. SELBY, "How Microsoft Builds Software," *Communications of the ACM* **40** (June 1997), pp. 53–61.

[Fowler et al., 1999] M. FOWLER WITH K. BECK, J. BRANT, W. OPDYKE, AND D. ROBERTS, *Refactoring: Improving the Design of Existing Code,* Addison-Wesley, Reading, MA, 1999.

[Gane, 1989] C. GANE, *Rapid System Development: Using Structured Techniques and Relational Technology,* Prentice Hall, Englewood Cliffs, NJ, 1989.

[Gilb, 1988] T. GILB, *Principles of Software Engineering Management,* Addison-Wesley, Wokingham, U.K., 1988.

[Henderson-Sellers and Edwards, 1990] B. HENDERSON-SELLERS AND J. M. EDWARDS, "The Object-Oriented Systems Life Cycle," *Communications of the ACM* **33** (September 1990), pp. 142–59.

[Honiden, Kotaka, and Kishimoto, 1993] S. HONIDEN, N. KOTAKA, AND Y. KISHIMOTO, "Formalizing Specification Modeling in OOA," *IEEE Software* **10** (January 1993), pp. 54–66.

[ISO/IEC 12207, 1995] "ISO/IEC 12207:1995, Information Technology—Software Life-Cycle Processes," International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Jacobson, Booch, and Rumbaugh, 1999] I. JACOBSON, G. BOOCH, AND J. RUMBAUGH, *The Unified Software Development Process,* Addison-Wesley, Reading, MA, 1999.

[Jones, 1994c] C. JONES, *Assessment and Control of Computer Risks,* Prentice Hall, Englewood Cliffs, NJ, 1994.

[Karolak, 1996] D. W. KAROLAK, *Software Engineering Risk Management,* IEEE Computer Society, Los Alamitos, CA, 1996.

[Keil, Cule, Lyytinen, and Schmidt, 1998] M. KEIL, P. E. CULE, K. LYYTINEN, AND R. C. SCHMIDT, "A Framework for Identifying Software Project Risks," *Communications of the ACM* **41** (November 1998), pp. 76–83.

[Landis et al., 1992] L. LANDIS, S. WALIGARA, F. MCGARRY, ET AL., "Recommended Approach to Software Development: Revision 3," Technical Report SEL-81-305, Software Engineering Laboratory, Greenbelt, MD, June 1992.

[Luqi and Royce, 1992] LUQI AND W. ROYCE, "Status Report: Computer-Aided Prototyping," *IEEE Software* **9** (November 1992), pp. 77–81.

[Mantei and Teorey, 1988] M. M. MANTEI AND T. J. TEOREY, "Cost/Benefit Analysis for Incorporating Human Factors in the Software Development Lifecycle," *Communications of the ACM* **31** (April 1988), pp. 428–39.

[McConnell, 1996] S. MCCONNELL, "Daily Build and Smoke Test," *IEEE Computer* **13** (July 1996), pp. 144, 143.

[Rajlich, 1994] V. RAJLICH, "Decomposition/Generalization Methodology for Object-Oriented Programming," *Journal of Systems and Software* **24** (February 1994), pp. 181–86.

[Rajlich and Bennett, 2000] V. RAJLICH AND K. H. BENNETT, "A Staged Model for the Software Life Cycle," *IEEE Computer* **33** (July 2000), pp. 66–71.

[Royce, 1970] W. W. ROYCE, "Managing the Development of Large Software Systems: Concepts and Techniques," *1970 WESCON Technical Papers, Western Electronic Show and Convention,* Los Angeles, August 1970, pp. A/1-1–A/1-9. Reprinted in *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, May 1989, pp. 328–38.

[Royce, 1998] W. ROYCE, *Software Project Management: A Unified Framework,* Addison-Wesley, Reading, MA, 1998.

[Spivey, 1992] J. M. SPIVEY, *The Z Notation: A Reference Manual,* Prentice Hall, New York, 1992.

[Williams, Kessler, Cunningham, and Jeffries, 2000] L. WILLIAMS, R. R. KESSLER, W. CUNNINGHAM, AND R. JEFFRIES, "Strengthening the Case for Pair Programming," *IEEE Software* **17** (July/August 2000), pp. 19–25.

제4장. 개 발 팀

충분한 자격이 있는 잘 숙련된 쏘프트웨어공학자들이 없다면 쏘프트웨어프로젝트는 실패의 운명에 처하게 된다. 그러나 능력 있는 사람들을 얻기는 쉽지 않다. 즉 개발팀은 팀의 성원들이 다른 성원들과 협력하여 능률적으로 작업할수 있도록 조직되여야 한다. 팀의 조직과 관련한 문제가 이 장의 주제이다.

4. 1. 개발팀의 조직

대부분의 제품들은 너무 커서 주어 진 제약된 시간안에 한명의 쏘프트웨어전문가가 완성할수 없다. 결과 제품은 팀으로 조직된 전문가들의 그룹에 맡겨 져야 한다. 실례로 명세작성단계를 고찰해 보자. 목표제품을 2개월안에 개발하기 위하여서는 명세작성관리자의 지도밑에 팀으로 조직된 세명의 명세작성전문가들에게 과제를 맡겨야 한다. 이와 류사하게 설계과제는 설계팀성원들에게 나누어 주어야 한다.

이제 한 사람이 1년동안 작성해야 할 코드를 포함하고 있는 제품이 3개월안에 작성되여야 한다고 가정하자. 해답은 명료하다. 즉 만일 한명의 프로그람작성자가 1년동안에 제품을 개발할수 있다면 4명의 프로그람작성자는 3개월동안에 해낼수 있다.

물론 이렇게 일하지는 못한다. 실천적으로는 4명의 프로그람작성자가 개발해도 거의 1년이 걸리고 결과적인 제품의 품질도 한명의 프로그람작성자가 혼자서 제품을 개발하는 것보다 더 떨어 질수 있다. 그 리유는 일부 과제는 분할될수 있지만 다른 과제들은 개별적으로 진행되여야 한다는것이다. 실례로 만일 한명의 농장원이 10일동안에 딸기를 수확할수 있다면 같은 량의 딸기를 10명이 하루동안에 수확할수 있다. 다른 한편 한명의 녀성이 9달동안에 아이를 만들수 있지만 9명의 녀성이 한달동안에 아이를 만들수는 없다.

달리 말하면 딸기따기와 같은 과제는 완전히 분할할수 있지만 아이를 낳는것과 같은 다른 문제들은 분할할수 없다. 아이를 낳는것과는 달리 팀성원들에게 코드작성을 분배함으로써 팀성원들에게 실현해야 할 과제를 분할해 줄수 있다. 그러나 팀프로그람작성은 팀성원들이 어떤 의미 있고 효과적인 방식으로 서로 대화해야 한다는데서 딸기따기와는 다르다. 실례로 제인과 네드가 두개의 모듈 m1와 m2를 코드작성해야 한다고 하자. 많은 일들이 잘못 진행될수 있다. 실례로 제인과 네드는 모두 m1을 코드작성하고 m2는 무시할수도 있다. 또는 제인은 m1을 코드작성하고 네드는 m2를 코드작성할수도 있다. 그러나 m1이 m2를 호출할 때 그것은 4개의 인수를 넘겨 주고 네드는 5개의 인수를 요구하는 방법으로 m2를 코드작성한다. 또는 m1과 m2에서 인수들의 순서가 다를수 있다. 또는 순서는 같지만 자료형이 약간 차이날수 있다. 이러한 문제들은 개발기업체를 통해 보급되지 않은 설계단계에서 만들어 진 결정에 의해서 생겨 나게 된다. 이런 문제는 팀프로그람작성자들의 기술적인 능력과는 아무런 관련도 없다. 팀의 조직은 관리상의 문제이다. 즉 관

리자측은 매 팀이 고도로 능률적이도록 프로그람작성팀을 조직해야 한다.

쏘프트웨어의 팀조직에서 제기될수 있는 다른 류형의 난점은 그림 4-1에 보여 주었다.

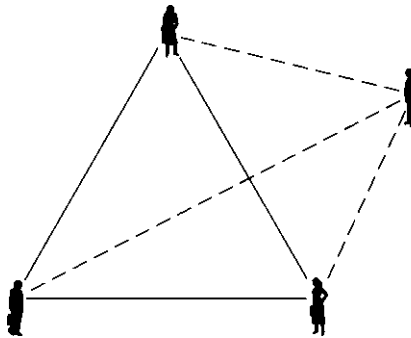



그림 4-1. 세명의 콤퓨터전문가들사이의 통신경로(실선), 네번째 전문가가 그들과 련결되여 있는 경우(점선)

프로젝트에 대하여 작업하는 세명의 콤퓨터전문가들사이에는 세개의 통신선로가 존재한다. 이제 작업이 잘못되여 최종기간이 거의 다가오고 있으나 과제가 아직 완성되지 않았다고 하자. 명백한것은 팀에 네번째 전문가를 보충하는것이다. 그러나 네번째 전문가가 팀에 보충될 때 해야 할 첫번째 일은 다른 세명이 어느 날까지 무엇을 완성해야 하고 아직 무엇이 완성되지 못했는가를 자세히 설명해 주는것이다. 달리 말하면 뒤늦게 쏘프트웨어프로젝트에 성원을 보충하는것 역시 개발이 더 늦어 지게 한다는것이다. 이 원리를 프레드 브룩스가 OS/360의 개발을 하면서 그것을 발견한후부터 브룩스의 법칙(Brooks’s Law)으로 불리워 졌다[Brooks, 1975].

큰 기업체에서 팀들은 프로그람작성자들이 독립적으로 작업하는 동안 쏘프트웨어생산의 매 단계에서 특히 실현단계에서 리용되게 된다. 따라서 실현단계는 몇명의 콤퓨터전문가들에게 과제를 분할하는 최초의 단계로 된다. 보다 작은 일부 기업체들에서는 하나의 개별적인 팀들에 요구사항확정과 명세작성, 설계를 맡길수 있으며 그다음 2 또는 3명의 프로그람작성자들로 이루어 진 팀에 의해서 실현단계가 진행된다. 팀들은 실현단계에서 가장 과중하게 리용되기때문에 팀조직문제는 실현단계에서 가장 날카롭게 제기된다. 따라서 이 장의 나머지부분에서 비록 문제점들과 그 문제점에 대한 해결이 다른 모든 단계들에서 마찬가지로 적용될수 있다고 해도 팀의 조직문제는 실현단계의 범위내에서 제시된다.

프로그람작성팀조직에 대한 두개의 극단적인 방법들이 있는데 하나는 민주적인 팀이고 다른 하나는 책임프로그람작성자팀이다. 여기서 취한 방법은 매 방법들을 서술하고 그 우단점을 강조하며 그다음 이 두 극단의 가장 좋은 특성들을 병합한 하나의 프로그람작성팀을 조직하는 다른 방법들을 제기하는것이다.

4. 2. 민주적팀방법

민주적팀조직에 대하여서는 1971년에 와인버그가 처음으로 서술하였다[Weinberg, 1971]. 민주적팀의 밑바탕에 놓여 있는 기초적인 개념은 주관이 없는 프로그람작성(*egoless programming*)이다. 와인버그는 프로그람작성자들은 자기들이 작성한 코드에 매우 애착을 가질수 있다고 강조하였다. 때때로 그들은 자기들이 작성한 모듈들에 자기들의 이름을 달아 부르고 있다. 즉 그들은 자기들이 작성한 모듈을 자기자신의 확장으로 생각한다. 이로부터 생기는 난점은 모듈을 자기자신의 확장이나 주관으로 보는 프로그람작성자들은 확실히 자기들의 코드에서 모든 오유를 찾아 내려고 하지 않는다는것이다. 만일에 오유가 존재한다면 그것을 바그(*bug*)라고 부르는데 그것은 승인없이 코드에 기여 들어 와서 코드가 침입에 대처하여 보다 더 열정적으로 감시할 때에만 막을수 있는 그런 벌레와 같다. 이러한 립장은 몇년전에 쏘프트웨어가 아직 착공카드로 입력되고 있을 때 그 벌레쫓개(*Shoo-Bug*)라고 부르는 공기식분무기를 파는것으로써 재미 있게 풍자되였다. 벌레쫓개로 프로그람카드를 분무하는것은 그 어떤 벌레도 코드에 기생할수 없다고 담보한다는것을 설명하고 있다. 프로그람작성자들이 자기자신의 코드와 매우 밀접하게 련관되여 있는데서 생기는 문제에 대한 와인버그의 해결방안은 주관이 없는 프로그람작성이다. 사회적인 환경은 재구성되여야 하며 따라서 프로그람작성자들에 대한 평가도 달라져야 한다. 모든 프로그람작성자들은 자기들의 코드에서 오유를 찾도록 팀의 다른 성원들을 부추겨야 한다. 오유가 존재한다는것을 그 어떤 나쁜것으로 간주될것이 아니라 일반적이고 허용할수 있는 사건으로서 간주하여야 한다. 즉 심사자들이 취해야 할 태도는 코드작성에서 오유를 범하는 프로그람작성자에 대하여 비웃을것이 아니라 충고를 받았다고 감사히 여겨야 한다는것이다.

10명이내의 주관이 없는 프로그람작성자들로 이루어 진 그룹은 민주적인 팀을 구성한다. 와인버그는 이러한 팀과 함께 일하는 경우에 관리상 곤난하다고 경고하였다. 그러면 관리상 성공하는 길을 생각해 보자. 프로그람작성자가 경영자의 지위에 등용되였을 때에 그의 동료프로그람작성자들은 등용되지 않고 다음단계의 등용에서 보다 더높이 올라 서려고 노력해야 한다. 이와 대비적으로 민주적인 팀은 지도하는 사람도 없고 다음수준으로 승급하려고 애 쓰는 그런 프로그람작성자도 없으며 다같이 공동의 목적을 달성하기 위하여 일하는 하나의 그룹이다. 중요한것은 팀이 단합되고 호상존중하는것이다.

와인버그는 민주적인 팀에 의해서 우수한 제품이 개발될수 있다고 하였다. 관리자측은 팀의 명목상 관리자(정의에 의하면 민주적인 팀에는 지도하는 사람이 없다.)에게 상금을 주기로 결정하였다. 명목상 관리자는 그것을 개인적으로 받아 들이는것을 거절하면서 팀의 모든 성원들에게 꼭같이 나누어 주어야 한다고 말했다. 관리자는 그가 더 많은 돈을 빼내려고 하고 있고 팀(특히는 명목상관리자이다.)은 오히려 이단적인 생각을 가지고 있다고 보았다. 관리자는 명목상의 관리자에게 돈을 받으라고 강요하였는데 그때 그는 팀의 성원들에게 꼭같이 나누어 주었다. 그다음에 전체적인 팀은 다시 다른 회사와 하나의 팀으로서 서명하고 결합한다.

민주적인 팀의 우점과 결함을 보기로 하자.

4. 2. 1. 민주적팀에 대한 분석

민주적팀방법의 중요한 우점은 오유찾기에서 긍정적인 태도를 취한다는것이다. 오유를 더 많이 찾으면 찾을수록 민주적인 팀의 성원들은 더 좋아 한다. 이러한 긍정적인 태도는 보다 빨리 오유를 발견하도록 하며 이로부터 코드의 질이 높아 진다. 그러나 여기에는 일부 중요한 문제점들이 있다. 이미 강조한바와 같이 관리자들은 주관이 없는 프로그람작성을 받아 들여야 한다는 난점을 가지고 있다. 이밖에 15년의 경험을 가진 프로그람작성자는 동료프로그람작성자들 특히는 초학도들에 의하여 코드가 평가되는것을 싫어한다.

와인버그는 주관이 없는 팀들이 자원적으로 형성되며 외부로부터 강요될수는 없다고 생각했다. 민주적인 프로그람작성팀에 대하여 실험조사가 거의나 진행되지 않았지만 와인버그의 경험은 민주적인 개발팀이 아주 능률적이라는것이다. 만테이(Mantei)는 특히 프로그람작성팀[Mantei, 1981]에 대해서가 아니라 일반적인 그룹기업체의 리론과 그에 대한 경험에 기초한 론증을 리용하여 민주적인 팀조직을 분석하였다. 그는 중앙집권적이 아닌 그룹은 문제가 어려울 때 가장 훌륭히 일하며 민주적인 개발팀은 연구환경에서 기능을 잘 수행한다는것을 제기하였다. 경험은 해결하기 힘든 문제가 있을 때에는 공업적인 환경에서 민주적인 팀이 역시 잘 일한다는것이다. 많은 경우에 프로그람작성자들은 연구경험을 가진 콤퓨터전문가들가운데서 자원적으로 제기하여 들어 온 민주적인 팀의 성원들이다. 그러나 일단 과제가 해결하기 어려운 실현으로 귀착되게 되면 팀은 다음장에서 서술되는 책임프로그람작성자팀과 같은 보다 계층적인 형식으로서 재조직되여야 한다.

4. 3. 고전적책임프로그람작성자팀방법

그림 4-2에 보여 준 6명으로 구성된 팀을 고찰해 보자.

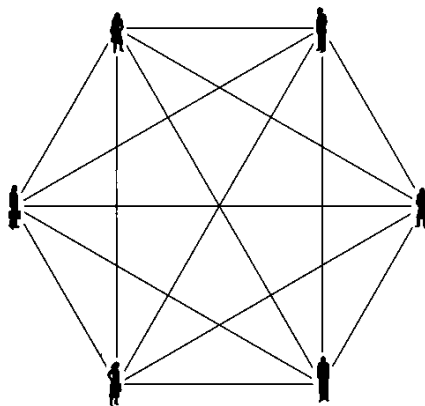



그림 4-2. 세명의 콤퓨터전문가들사이의 통신경로

15개의 2인통신선로가 존재한다. 사실상 2, 3, 4, 5, 6명으로 구성되는 그룹들의 총수는 57개이다. 이러한 통신선로의 다양성은 그림 4-2에서와 같이 구성된 6명으로 이루어진 팀이 36명이 1개월동안 수행하는 작업을 6개월동안에 수행할수 없는 리유로 된다. 즉 둘 또는 그이상의 팀성원들이 한번에 모여서 토론하는데 많은 시간이 걸리게 된다. 그림 4-3에서 보여 준 6명으로 구성된 팀에 대하여 고찰해 보자. 거기에는 6명의 프로그람 작성자들이 있지만 통신선로는 5개만 있다.

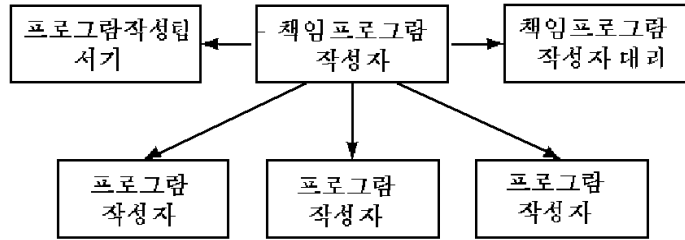



그림 4-3. 고전적책임프로그람작성자팀의 구조

이것은 지금 책임프로그람작성자(*chief programmer*)팀이라는 용어의 리면에 놓여 있는 기초적인 개념으로 된다. 이와 련관된 착상을 브룩스가 제기하였는데 그는 수술을 지도하는 책임의사를 류추하였다[Brook 1975]. 책임의사는 다른 의사들과 마취사, 여러 간호원들의 조력을 받는다. 이밖에 필요한 경우 심장전문의사나 콩팔전문의사들과 같은 다른 부분의 전문가들을 리용한다. 이러한 류추는 책임프로그람작성자팀에 대한 두개의 중요한 측면을 강조하고 있다. 첫번째는 전문화(*specialization*)이다. 즉 팀의 매 성원들을 자기가 숙련된 그러한 과제들만 수행한다. 두번째는 계층성(*hierarchy*)이다. 즉 책임의사는 팀의 모든 성원들의 행동을 지휘하며 수술의 모든 측면에 대하여 책임을 진다.

책임프로그람작성자팀의 개념은 밀스(Mills)에 의해서 형식화되였다[Baker, 1972]. 약 30년전에 베이커가 서술한 고전적책임프로그람작성자팀을 그림 4-3에 보여 주었다. 그것은 책임프로그람작성자대리, 프로그람작성팀서기, 한명부터 세명까지의 프로그람작성자들의 방조를 받는 책임프로그람작성자로 구성되여 있다. 필요할 때 팀은 법률이나 재정적인 사업 또는 조작체계명령을 해당한 그 시대의 대형콤퓨터에 보내주는데 리용되는 일감조종언어(JCL)명령과 같은 다른 령역의 전문가들로부터 방조를 받게 된다. 책임프로그람작성자는 성공적인 관리자인 동시에 구성방식설계와 중요하거나 복잡한 코드부분들을 작성한 고도로 숙련된 프로그람작성자이기도 하였다. 다른 팀 성원들은 책임프로그람작성자의 지도밑에 상세설계와 코드작성에 종사한다. 그림 4-3에서 보여 준바와 같이 프로그람작성자들사이에는 통신선로가 없다. 즉 모든 대화적문제들은 책임프로그람작성자에 의해서 조종된다. 마지막으로 책임프로그람작성자는 다른 팀성원들의 작업을 검토한다. 왜냐하면 책임프로그람작성자는 개인적으로 코드의 모든 행들에 대하여 책임을 지고 있기때문이다.

그러나 책임프로그람작성자도 사람인만큼 앓을수도 있고 사고가 날수도 있으며 직업

이 변경될수도 있기때문에 책임프로그람작성자대리의 지위가 필요하게 된다. 따라서 책임프로그람작성자대리는 모든 측면에서 책임프로그람작성자만큼 자격이 충분해야 하며 프로젝트에 대하여 많은것을 알고 있어야 한다. 이밖에 책임프로그람작성자가 구성방식설계에 집중할수 있도록 하기 위하여 책임프로그람작성자대리는 검은통시험실례계획작성(14.7)과 설계공정과는 독립인 다른 과제들을 수행하게 된다.

서기라는 단어는 많은 의미를 가지고 있다. 서기는 전화를 받거나 편지를 쓰는 등의 업무를 수행함으로써 책임자의 바쁜 사업을 도와 준다. 그러나 국가적인물들에 대하여 생각할 때에는 서기를 고급사무원들중의 한 성원으로 볼수 있다. 프로그람작성팀서기는 서기일을 겸임하는 조수인것이 아니라 고도로 숙련되고 높은 보수를 받는 책임프로그람작성자팀의 중심인물로 된다. 프로그람작성팀서기는 프로젝트제품서고와 프로젝트의 문서를 유지정비할 책임을 지고 있다. 여기에는 또한 원천코드목록작성과 JCL, 시험자료들이 포함된다. 프로그람작성자는 자기들이 작성한 원천코드를 서기에게 주는데 서기는 그것을 기계가독형식으로 변환하고 콤파일, 련결, 적재, 실행하며 시험실례들을 실행할 책임을 지게 된다. 따라서 프로그람작성자는 프로그람작성이외의 다른 일은 하지 않는다. 기타 다른 모든 측면의 작업은 프로그람작성팀서기가 조종한다(프로그람작성팀서기는 프로젝트제품생산서고를 유지관리하기때문에 일부 기업체들에서는 그들을 사서라고 부른다.).

여기서 서술된 내용들은 밀스와 베이커가 내놓은 착상이라는것을 상기하자. 1971년까지 거슬러 올라 가보면 그때는 아직도 착공기가 널리 리용되고 있었다. 코드작성은 오늘 더는 그런 방법으로 진행되지 않는다. 이제는 프로그람작성자가 말단이나 작업콤퓨터상에서 코드를 입력하고 편집하고 시험하는 등의 일을 한다. 현대의 고전적책임프로그람작성자팀에 대해서는 4.4에서 서술한다.

4. 3. 1. 뉴욕타임스프로젝트

책임프로그람작성자팀개념은 1971년에 IBM회사가 뉴욕타임스의 신문기사자료철(《자료부》)을 자동화하기 위하여 처음으로 리용하였다. 신문기사파일에는 뉴욕타임스와 다른 출판사들에서 출판한 개괄자료들과 전체 기사들이 포함되여 있다. 보고서작성자와 편집부의 기타 성원들은 이 정보창고를 참고원천으로서 리용한다.

이 프로젝트의 실상은 사람들을 몹시 놀라게 하고 있다. 실례로 8만 3천여개의 코드행(LOC)이 11명의 1년동안의 노력에 맞먹는 22개월동안에 작성되였다. 첫해후에 다만 12,000LOC로 구성된 파일관리체계가 작성되였다. 대부분의 코드는 마지막 6개월동안에 작성되였다. 첫 5주동안의 인수시험기간에는 다만 21개의 오유가 발견되였다. 그이후의 25개 오유는 제품이 가동한 첫해에 발견되였다. 원리적으로 프로그람작성자들은 평균 하나의 오유를 발견하며 한사람당 1년에 10,000LOC의 코드를 작성한다. 파일유지정비체계는 코드작성이 완성된 다음 한주일후에 배포되여 간단한 오유가 발견되기전까지 20개월동안 동작하였다. PL/1에서 보통 200～400행에 달하는 부분프로그람의 거의 절반이 첫번째의 콤파일에서 정정되였다[Baker, 1972].

그러나 이러한 공상적인 성공을 이룩한 다음에도 책임프로그람작성자팀개념에 대한 그 어떤 명백한 요청은 제기되지 않았다. 많은 성공적인 프로젝트들이 책임프로그람작성자팀을 리용하여 개발되였으나 그것은 뉴욕타임스프로젝트에서처럼 그렇게 인상적이지는 못했다. 무엇때문에 뉴욕타임스프로젝트가 그처럼 성공적인가? 무엇때문에 다른 프로젝트들에서는 류사한 결과가 얻어 지지 않는가?

그 첫번째 리유는 그것이 IBM의 명성과 관련된 프로젝트였다는것이다. 그것은 IBM에서 개발한 언어인 PL/1에 있어서 처음으로 되는 실제적인 시험이였다. 뛰여 난 쏘프트웨어전문가들에게 잘 알려 진 IBM은 자기 부문에서 가장 우수하다고 할수 있는 사람들을 망라하여 팀을 구성하였다. 둘째로 기술적후원이 아주 강했다. PL/1콤파일러작성자들은 가능한 모든 방법으로 프로그람작성자들을 직접 방조하였다. 그리고 JCL전문가들은 일감조종언어로써 방조하였다. 세번째 리유는 책임프로그람작성자 에프. 테리 베이커의 우수한 전문지식에 있다. 그는 현재 최고급프로그람작성자라고 불리우고 있다. 여기서 최고급프로그람작성자는 보통 훌륭한 프로그람작성자들에 비하여 4배 또는 5배의 결과를 내놓을수 있는 프로그람작성자들이다. 이밖에 베이커는 뛰여 난 관리자, 책임자였으며 그의 기능과 열정, 성격은 프로젝트의 성공의 비결로 되였다.

만일 책임프로그람작성자가 충분한 자격을 가지고 있다면 책임프로그람작성자팀조직은 잘 운영된다. 비록 뉴욕타임스프로젝트에서 이룩한 주목할만한 성과는 되풀이되지는 않았지만 많은 프로젝트들에 책임프로그람작성자방법의 변종들이 리용되였다. 이 방법의 변종이 리용된 리유는 문헌 [Baker, 1972]에 서술된바와 같이 고전적책임프로그람작성자팀이 많은 면에서 비현실적이기때문이다.

4. 3. 2. 고전적책임프로그람작성자팀방법의 비현실성

프로그람작성자와 성공적인 경영자의 결합체인 책임프로그람작성자를 고찰하자. 이러한 사람들은 찾아 보기 힘들다. 즉 성공적인 관리측면은 물론 고도로 숙련되지 않을수 있는데 책임프로그람작성자의 업무사항들은 이 두가지 능력을 모두 요구한다. 고도로 숙련된 프로그람작성자로서의 자질은 성공적인 관리자로서의 자질과 차이가 있으며 따라서 책임프로그람작성자를 찾아 낼 기회는 적다. 그러나 만일 책임프로그람작성자를 찾아 내기 힘들다면 책임프로그람작성자대리를 찾아 내기는 더욱 힘들다. 결국 책임프로그람작성자대리는 책임프로그람작성자만큼 훌륭하다고 생각되지만 책임프로그람작성자에게서 무엇인가 일어 나기를 기다리는 동안에는 뒤전에 앉아서 보다 낮은 로임을 받지 않으면 안된다. 고급프로그람작성자나 고급관리자들은 거의나 이러한 역할을 수행하기를 바라지 않는다.

프로그람작성팀서기 역시 찾아 내기가 힘들다. 쏘프트웨어전문가들은 문서작업을 하기 싫어 하는것으로 유명한데 프로그람작성팀서기는 하루종일 아무것도 하지 않고 문서작업이외에는 그 어떤 작업도 하지 않는것으로 생각하고 있다.

이리하여 베이커가 주장한것과 같이 적어도 책임프로그람작성자팀들은 실천적으로 비현실적이다. 민주적인 팀도 역시 다른 리유로 하여 비현실적이다. 더우기 두가지 팀방

법들은 모두 실현단계에서 120명은 물론 20명의 프로그람작성자를 요구하는 제품조차 조종할수 없을것 같다. 민주적인 팀과 책임프로그람작성자팀의 우점을 리용하여 보다 더 큰 제품이 실현될수 있도록 하는 프로그람작성팀을 조직하는것이 필요하다.

4. 4. 책임프로그람작성자팀과 민주적팀의 초월

민주적인 팀은 주요한 우점을 가지고 있다. 즉 오유를 찾는데서 긍정적인 태도를 가지고 있다는것이다. 많은 기업체들은 코드검토(6.2)와 관련하여 잠재적인 과오를 범하면서 책임프로그람작성자팀을 리용한다. 책임프로그람작성자는 코드의 모든 행에 대하여 개인적으로 책임을 져야 하며 따라서 모든 코드를 검토하는 동안 참석해야 한다. 그러나 책임프로그람작성자는 역시 6장에서 설명하게 되는바와 같이 관리자이며 검토는 그 어떤 성능평가에 리용되지 말아야 한다. 그리고 책임프로그람작성자 역시 팀성원들에 대한 초기의 평가를 책임진 관리자이기때문에 그자신이 코드를 검토하는데 참가한다는것은 좋은 방책으로 되지 않는다.

이러한 모순으로부터의 출로는 책임프로그람작성자로부터 여러가지 관리상의 기능을 떼여 내는것이다. 결국 아주 기능이 높은 프로그람작성자이면서도 성공적인 관리자인 한명의 개별적인 사람을 찾아 내는것이 어렵다는것은 이미 앞에서 지적되였다. 대신에 책임프로그람작성자는 팀활동의 기술적인 측면을 책임진 팀책임자와 모든 비기술적인 관리상의 결심을 내리는데서 책임을 지는 팀관리자 두명으로 교체되여야 한다. 결과적인 팀의 구조를 그림 4-4에 보여 주었다.

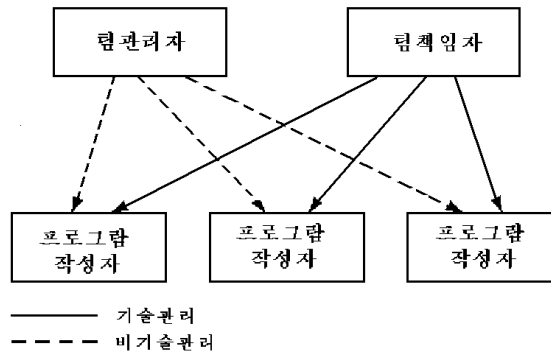



그림 4-4. 현대프로그람작성자팀구조

이러한 조직적인 구조가 종업원들이 한명의 관리자이외의 다른 사람들에게 보고하지 않는다는 관리상 원칙을 위반하지 않도록 하는것이 중요하다. 책임의 범위를 명백히 밝혀야 한다. 팀책임자는 오직 기술적인 관리에 대해서만 책임을 진다. 이리하여 예산작성과 법률상 문제는 팀책임자가 조종하지 않으며 또한 성능평가 역시 마찬가지이다. 다

른 한편 팀책임자는 기술적인 문제점들에 대하여 혼자서 책임을 지고 있다. 그러므로 팀관리자는 제품이 4주안에 배포될것이라는 계약을 체결할 권리가 없다. 즉 그런 종류의 계약은 팀책임자가 해야 한다. 팀책임자는 자연이 모든 코드검토에 참가한다. 결국 그는 코드의 모든 측면에 대하여 혼자서 책임을 진다. 동시에 팀관리자에게는 검토가 허락되지 않는다. 왜냐하면 프로그람작성자의 성능평가가 팀관리자의 기능이기때문이다. 대신에 팀관리자는 정기적으로 진행되는 팀회의를 하는 동안에 모든 팀성원들의 기술기능에 대하여 알게 된다.

실현이 시작되기전에 팀관리자와 팀책임자의 책임이라고 볼수 있는 범위를 정확히 확정하는것이 중요하다. 실례로 휴가문제를 고찰하자. 휴가가 비기술적인 문제이기때문에 팀관리자가 휴가신청을 승인했는데 최종기한이 다가와서 팀책임자가 휴가신청을 거부하는 그런 상황이 일어 날수 있다. 이 문제와 련관된 문제의 해결방도는 팀관리자와 팀책임자들이 모두 자기들의 책임이라고 생각하는 령역들에 주목하면서 더잘 관리할수 있도록 대책을 작성하는것이다.

보다 더 큰 프로젝트에 대해서는 어떠한가? 이에 대한 방법은 그림 4-5에서 보여 준바와 같이 확장될수 있는데 여기서는 기술적이며 관리적인 조직구조를 보여 주고 있다. 그리고 비기술적인 측면도 류사하게 조직화되였다.

제품의 전반적인 실현은 프로젝트책임자의 의도에 따라 진행된다. 프로그람작성자는 자기들의 팀책임자에게 보고하고 팀책임자는 프로젝트책임자에게 보고한다. 지어 보다 큰 제품에 대해서는 보충적인 준위를 계층적으로 추가할수 있다. 민주적팀과 책임프로그람작성자팀들의 좋은 특성들을 모두 살리는 또 다른 방법은 적당한 곳에서 결심채택공정을 분산시키는것이다. 결과적인 통신선로를 그림 4-6에 보여 주었다.

이러한 구조도식은 민주적방법이 적당한 그런 종류의 문제들에서 쓸모가 있다. 즉 어떤 연구환경이나 또는 문제해결을 위하여 어려운 문제들이 협동적인 성격을 띤 그룹호상작용을 요구할 때마다 쓸모가 있게 된다. 분산화에도 불구하고 준위사이의 화살표는 프로그람작성자들이 혼란에로 이끌어 갈수 있는 프로젝트책임자들에게 지시하도록 허용하면서 여전히 아래로 향하고 있다. 그러나 프로그람작성자팀에 관한 문제, 프로그람작성팀조직문제 나아가서 다른 모든 단계들에서의 팀조직문제에 대해서는 그 어떤 해결책도 없다. 팀을 조직하는 최량방도는 개발해야 할 제품과 여러가지 팀구조에 대한 이전의 경험, 기업체책임자의 전도에 의존한다. 실례로 고급관리자측이 분산적인 결심채택에 동의하지 않으면 그때는 실현되지 않는다. 그러나 유감스럽게도 쏘프트웨어개발팀조직에 대해서는 많이 연구되지 않았으며 일반적으로 수용할수 있는 대부분의 연구들은 보통 쏘프트웨어개발팀에 대한 연구가 아니라 그룹의 움직임에 관한 연구에 기초하고 있다. 팀조직에 관한 실험적인 결과들이 쏘프트웨어산업에서 얻어 지기전에는 특정한 제품에 대하여 최량인 팀의 조직을 결정하기가 쉽지 않을것이다.

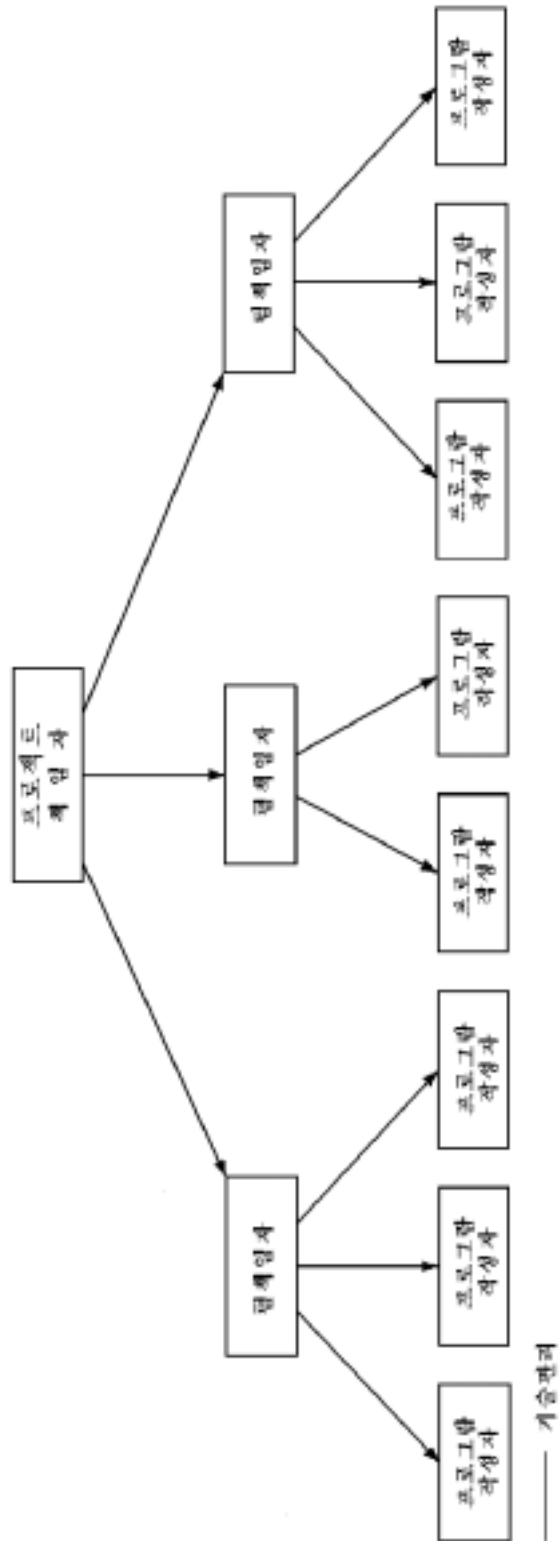



그림 4-5. 큰 규모 프로젝트를 위한 기술관리조직구조

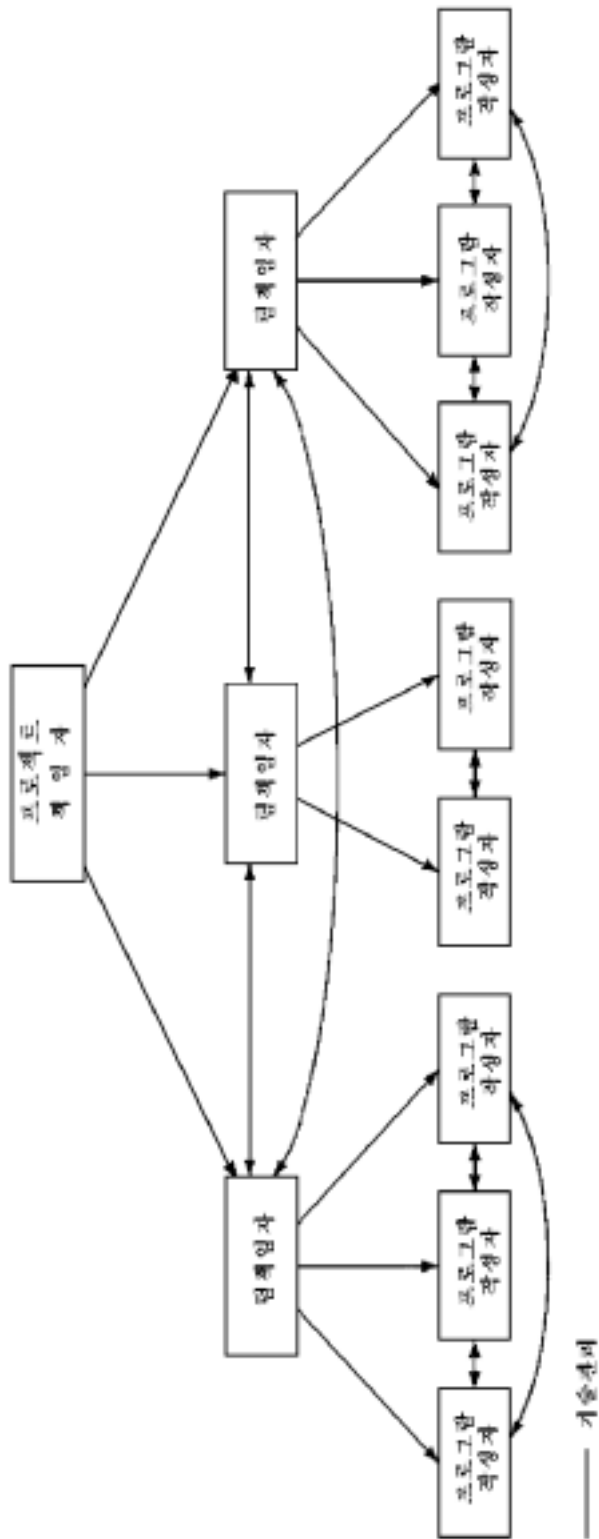



그림 4-6. 기술관리의 통신경로를 보여 주는 그림 4.5의 팀조직에 관한 분산결심채택

4. 5. 동기 및 안정화팀

팀조직에 대한 또 다른 방법에는 Microsoft에서 리용된 동기 및 안정화팀방법이 있다 [Cusumano and Selby, 1997]. Microsoft는 큰 제품을 개발한다. 실례로 윈도우즈 2000은 3천만개이상의 행을 가진 코드로 구성되여 있으며 3천명이 넘는 프로그람작성자들과 시험자들에 의해서 개발되였다[Business Week Online, 1999]. 팀조직은 이러한 규모의 크기를 가진 제품을 성공적으로 개발하기 위한 하나의 중요한 측면으로 된다.

동기 및 안정화생명주기모형은 3.6에서 서술되였다. 이 모형이 성공한것은 기본팀조직방법의 결과이다. 동기 및 안정화모형의 셋 또는 네개의 련속적인 매 구조물들은 프로그람관리자가 이끌며 3~8명의 개발자들과 그들과 일 대 일로 작업을 진행하는 3~8명의 시험자들로 구성된 여러개의 자그마한 병렬팀들에 의해서 구축된다. 팀은 전반적인 과제에 대한 명세서를 받게 된다. 개별적인 팀성원들은 다음에 그 과제들에서 자기들의 몫을 자유롭게 설계하고 실현하게 된다. 이것이 해커에 의해서 일어 나는 혼란에로 재빨리 넘어 가지 않는 리유는 매일 진행하는 동기화단계에 있다. 즉 부분적으로 완성된 구성요소는 매일 시험되고 수정된다. 이리하여 지어 매 사람들의 창조성과 자률성이 발휘된다고 할지라도 개별적인 구성부분은 항상 함께 동작한다.

한편 이러한 방법의 우점은 개별적인 프로그람작성자들이 어떤 민주적팀의 하나의 특성인 창조적이고 혁신적인 사람으로 되도록 고무한다는것이다. 다른 한편 매일 진행하는 동기화단계는 수백명의 개발자들이 공동의 목적을 위하여 책임프로그람작성자팀에 고유한 정보전달과 공동작용을 요구하지 않고 함께 일하고 있다는것을 담보한다(그림 4-3).

마이크로쏘프트개발자들은 매우 적은 규칙을 준수하여야 하는데 그중의 하나는 그들이 일별동기화를 보장하기 위하여 제품자료기지에 자기들의 코드를 넣어 보관하는 시간을 엄격히 준수해야 한다는것이다. 쿠수마노와 쎌비는 이것은 아이들에게 하루종일 그들이 하고 싶은 일을 할수 있으나 저녁 9시에는 잠을 자야 한다고 말해 주는것과 같다고 보았다[Cusumano and Selby, 1997]. 또 다른 하나의 규칙은 만일 개발자들의 코드가 제품이 일별 동기화를 위하여 콤파일되는것을 막기 위해서는 팀의 나머지성원들이 그날작업을 시험하고 오유검사를 하고 수정할수 있도록 문제가 즉시 수정되여야 한다는것이다.

동기 및 안정화모형의 리용과 그와 련관된 팀조직이 기타 다른 모든 기업체들이 마이크로쏘프트와 같이 성공할수 있다는것을 담보해 줄것인가? 이것은 전혀 그렇지 않다. 마이크로쏘프트회사는 동기 및 안정화모형이상의 수준에 있다. 마이크로쏘프트회사는 그룹적기질을 가지고 있는 뛰여 난 관리자들과 쏘프트웨어개발자들로 이루어 진 기업체이다. 단순히 동기 및 안정화모형만을 리용하는것은 기업체를 또 다른 하나의 마이크로쏘프트회사로 전환시키기는 기적을 가져 오지 못할것이다. 동시에 다른 기업체들에서 리용하는 모형들의 여러가지 특성을 리용하는것은 공정을 개선하도록 해준다. 다른 한편 동기 및 안정화모형은 해커들의 그룹이 큰 제품을 개발하도록 하는 간단한 방법으로 된다

는것이 제기되였다. 앞부분의 마감에서 언급한바와 같이 최량인 프로그람작성자팀조직과 관련하여 결론을 끌어 내기전에 경험적인 자료가 필요하다.

4. 6. 최종적프로그람작성팀

3.5에서 최종적프로그람작성팀에 대하여 개관하였다[Beck, 1999]. 이 절에서 최종적프로그람작성(XP)이 리용될 때 팀이 어떻게 조직되는가를 서술한다.

XP의 약간 례외적인 특성은 모든 코드가 하나의 콤퓨터를 공유하고 있는 두명의 프로그람작성자들로 구성된 팀에 의해서 작성된다는것이다. 이것을 2인프로그람작성이라고 부른다[Williams, Kessler, Cunningham, and Jeffries, 2000]. 여러가지 리유로 하여 이 방법을 리용한다.

1. 3.5에서 설명한바와 같이 프로그람작성자들은 먼저 시험실례를 작성하고 그다음에는 코드(과제)의 해당한 부분을 실현한다. 6.2에서 설명한바와 같이 프로그람작성자들이 자기들의 코드를 시험하도록 하는것은 좋은 방책이 아니다. XP는 팀의 한 프로그람작성자가 한 과제에 대한 시험실례를 작성하고 다른 프로그람작성자가 그와 결합하여 이 시험실례들을 리용하여 코드를 실현하도록 함으로써 이 문제를 에돌아 가게 한다.
2. 보다 습관적인 생명주기모형에서는 개발자들이 어떤 프로젝트를 포기할 때 그 개발자들이 축적한 모든 지식도 역시 버려 진다. 특히는 개발자들이 작업하고 있는 쏘프트웨어는 아직 문서로 작성되여 있지 않을수도 있고 처음부터 다시 개발하여야 할수도 된다. 대조적으로 만일 2인프로그람작성팀의 한명이 떠나가면 다른 성원은 새로운 2인프로그람작성자와 함께 쏘프트웨어의 같은 부분을 계속 개발할수 있는 충분히 지식을 가지고 있다. 더우기 시험실례들은 오유를 밝히는데 도움을 주며 새팀은 무분별한 변경을 만듦으로써 쏘프트웨어를 손상시키게 될것이다.
3. 2명이 밀접히 결합되여 일하는것은 경험이 어린 쏘프트웨어전문가들로 하여금 더 경험 있는 팀성원들의 기능을 획득할수 있게 한다.
4. 3.5에서 언급한바와 같이 여러가지 XP 2인팀들이 리용하는 모든 콤퓨터들은 큰 방가운데 함께 놓여야 한다. 이것은 주관이 없는 팀들의 긍정적인 특성인 코드에 대한 그룹소유를 촉발시킨다(4.2).

이리하여 두명의 프로그람작성자들이 같은 콤퓨터상에서 함께 작업할데 대한 착상이 약간 례외적인것 같다 하더라도 실천에서는 뚜렷한 우월성을 가진다. 이것으로 팀조직에 대한 론의는 결속된다. 여러가지 류형의 팀조직들의 비교를 그림 4-7에 주었다.

팀 조 직	우 점	약 점
민주적팀	오유탐색에서 긍정적인 태도와 높은 품질이다.	외부적으로 강요할수 없다.
고전적책임프로그람작성자팀	뉴욕타임스프로젝트에서 기본 성공하였다.	비실용적이다.
수정된 책임프로그람 작성자팀	많이 성공하였다.	뉴욕타임스프로젝트와 비교할만한 성과는 없다.
현대의 계층적프로그람 작성팀	팀관리자/팀책임자구조 책임프로그람작성자의 요구를 거절한다. 규모를 확장한다. 필요하면 분산을 지원한다.	팀관리자와 팀책임자의 책임한계가 명백하게 설정되여 있지 않으면 문제가 발생한다.
동기 및 안정화팀	창발성을 불러 일으킨다. 아주 많은 개발자들이 공동의 목적을 위하여 일할수 있게 한다.	이 방법을 마이크로쏘프트회사외에서 리용할수 있다는 증거는 없다.
최종프로그람작성팀	프로그람작성자는 자기의 코드를 시험하지 않는다. 한명의 프로그람작성자가 남아도 지식은 잃지 않는다.	여전히 그 효력에 대하여서는 증거가 매우 적다.

그림 4-7. 팀조직방법의 비교

요 약

팀조직에 관한 론의(4.1)는 처음에 민주적팀(4.2)과 고전적책임프로그람작성자팀(4.3)을 고찰하고 그다음 두 방법의 우점을 리용한 팀조직 (4.4)을 제기하였다. 동기 및 안정화팀(마이크로쏘프트가 리용한)은 4.5에서 설명되였다. 마지막으로 최종적프로그람작성(XP)팀이 4.6에서 론의된다.

보 충

팀조직에 관한 고전적인 연구는 문헌 [Weinberg, 1971; Baker, 1972; and Brooks, 1975]이다. 그 주제에 대한 보다 새로운 책은 [DeMarco and Lister, 1987]과 [Cusumano and Selby, 1995]에 포함되여 있다. 어떻게 팀조직이 발전해 왔는가 하는데 대한 흥미 있는 설명은 문헌 [Mackey, 1999]에서 찾아 볼수 있다. 팀의 조직과

관리에 관한 기사는 *Communications of the ACM* 1993년 10월호에서 찾아 볼수 있다. 문헌 [Royce, 1998]의 11장은 팀성원들이 놀게 되는 역할에 대한 유용한 정보들을 포함하고 있다.

동기 및 안정화팀은 문헌 [Cusumano and Selby, 1997]에서는 개괄하여 주었고 문헌 [Cusumano and Selby, 1995]에서 자세히 서술하였다. 동기 및 안정화팀에 대한 지식은 문헌 [McConnell, 1996]에서 얻을수 있다. 최종적프로그람작성은 [Beck, 1999]와 [Beck, 2000]에 서술되여 있다. 문헌 [Williams, Kessler, Cunningham, and Jeffries, 2000]은 최종적프로그람작성의 한 구성부분인 2인프로그람작성에 대한 실험을 서술하고 있다.

문 제

4.1. 로임지불계산프로젝트를 개발하는 팀을 어떻게 조직하겠는가? 최첨단군사통신쏘프트웨어를 개발하기 위한 팀은 어떻게 조직하겠는가? 대답을 설명하시오.

4.2. 당신은 방금 새로운 쏘프트웨어회사를 개설하였다. 모든 종업원들은 최근에 단과대학을 졸업하였다. 그들은 처음으로 프로그람을 작성하게 된다. 당신의 론리에서 민주적개발팀을 실현할수 있는가? 그렇다면 어떻게 결합하겠는가?

4.3. 어떤 대학생프로그람작성팀은 민주적팀으로 조직된다. 팀에 있는 대학생들에 대하여 무엇을 추론할수 있는가?

4.4. 어떤 대학생프로그람작성팀은 책임프로그람작성자팀으로 조직된다. 팀에 있는 대학생들에 대하여 무엇을 추론할수 있는가?

4.5. 큰 쏘프트웨어회사안에 있는 두개의 서로 다른 팀 TO1과 TO2를 비교하기 위하여 다음의 실험을 제기한다. 같은 쏘프트웨어제품이 두개의 서로 다른 팀에 의하여 개발되는데 하나는 TO1에 따라 조직되고 다른 하나는 TO2에 따라 조직된다. 회사는 매 팀이 제품을 개발하는데 약 18개월이 걸릴것이라고 평가하였다. 이 실험이 왜 비현실적이며 의미 있는 결과를 주지 못하는가에 관한 세가지 리유를 드시오.

4.6. 왜 최종적프로그람작성팀이 하나의 콤퓨터를 공유하는가?

4.7. 당신은 동기 및 안정화팀을 리용하는 회사에서 일하고 싶은가? 당신의 대답을 정당화하시오.

4.8. (과정안상 목표) 부록 1에 있는 브로드랜즈지역 아동병원제품을 개발하기 위하여서는 어떤 형태의 팀조직이 적합한가?

4.9. (쏘프트웨어공학독본) 교원이 문헌 [Cusumano and Selby, 1997]의 복제본을 배포할것이다. 어떤 형식의 책임프로그람작성자팀을 리용하고 있는 회사에 동기 및 안정화팀을 어떻게 도입하겠는가?

참 고 문 헌

[Baker, 1972] F. T. BAKER, "Chief Programmer Team Management of Production Programming," *IBM Systems Journal* **11** (No. 1, 1972), pp. 56–73.

[Beck, 1999] K. BECK, "Embracing Change with Extreme Programming," *IEEE Computer* **32** (October 1999), pp. 70–77.

[Beck, 2000] K. BECK, *Extreme Programming Explained: Embrace Change,* Addison Wesley Longman, Reading, MA, 2000.

[Brooks, 1975] F. P. BROOKS, JR., *The Mythical Man-Month: Essays in Software Engineering,* Addison-Wesley, Reading, MA, 1975. Twentieth Anniversary Edition, Addison-Wesley, Reading, MA, 1995.

[Business Week Online, 1999] Business Week Online, February 2, 1999, www.businessweek.com/1999/99_08/b3617025.htm.

[Cusumano and Selby, 1995] M. A. CUSUMANO AND R. W. SELBY, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People,* The Free Press/Simon and Schuster, New York, 1995.

[Cusumano and Selby, 1997] M. A. CUSUMANO AND R. W. SELBY, "How Microsoft Builds Software," *Communications of the ACM* **40** (June 1997), pp. 53–61.

[DeMarco and Lister, 1987] T. DEMARCO AND T. LISTER, *Peopleware: Productive Projects and Teams,* Dorset House, New York, 1987.

[Mackey, 1999] K. MACKEY, "Stages of Team Development," *IEEE Software* **16** (July/August 1999), pp. 90–91.

[Mantei, 1981] M. MANTEI, "The Effect of Programming Team Structures on Programming Tasks," *Communications of the ACM* **24** (March 1981), pp. 106–13.

[McConnell, 1996] S. MCCONNELL, "Daily Build and Smoke Test," *IEEE Computer* **13** (July 1996), pp. 144, 143.

[Royce, 1998] W. ROYCE, *Software Project Management: A Unified Framework,* Addison-Wesley, Reading, MA, 1998.

[Weinberg, 1971] G. M. WEINBERG, *The Psychology of Computer Programming,* Van Nostrand Reinhold, New York, 1971.

[Williams, Kessler, Cunningham, and Jeffries, 2000] L. WILLIAMS, R. R. KESSLER, W. CUNNINGHAM, AND R. JEFFRIES, "Strengthening the Case for Pair Programming," *IEEE Software* **17** (July/August 2000), pp. 19–25.

제5장. 거래되고 있는 도구

쏘프트웨어공학자들은 두가지 류형의 도구를 요구한다. 첫째로 계단식세련과 비용대 리득분석과 같은 쏘프트웨어개발에서 리용되는 분석도구들이다. 둘째로 쏘프트웨어도구 즉 쏘프트웨어를 개발하고 유지정비하는데서 쏘프트웨어공학자들의 팀을 도와 주는 제품들이 있는데 이것들을 보통 CASE도구라고 한다(CASE는 콤퓨터지원프로그람공학의 략어이다.). 이 장에서는 이런 두가지 류형의 거래도구를 론의한다. 즉 처음에는 리론적 도구를, 그다음에는 쏘프트웨어(CASE)도구를 론의한다. 계단식세련으로부터 시작하자.

5．1．계단식세련

계단식세련은 많은 쏘프트웨어공학기법들의 기초를 이루고 있는 문제해결기법이다. 그것은《중요한 문제점들에 집중하기 위하여 세부들에 대한 결정은 가능한껏 뒤로 미룬다.》는 의미로서 정의될수 있다. 이 책을 읽어 나가는 과정에 볼수 있는바와 같이 계단식세련은 여러가지 명세작성기법, 설계 및 실현기법, 지어 시험 및 통합기법의 기초를 이루고 있다. 계단식세련이 그렇게 중요하게 된 리유는 밀러의 법칙[Miller, 1956]때문인데 그것은 한번에 사람은 기껏해서 7±2개의 토막들(정보량들)에 집중할수 있다는것을 말해준다.

쏘프트웨어를 개발할 때에 제기되는 문제는 한번에 7개이상의 토막들에 집중해야 한다는것이다. 실례로 하나의 클라스는 보통 7개보다 훨씬 더 많은 속성과 방법을 가지고 있으며 의뢰자는 7개이상의 요구사항를 가지고 있다. 계단식세련은 쏘프트웨어공학자들로 하여금 현재의 개발단계와 가장 관련 있는 7개의 토막들에 집중할수 있도록 한다. 다음의 실례는 제품의 설계단계에서 어떻게 계단식세련이 리용되는가를 설명하고 있다.

5．1．1．계단식세련의 실례

여기서 취급하는 실례는 많은 응용령역에 있는 공통적인 조작들인 련속적인 주파일들을 갱신하는것을 포함하고 있다는 점에서 거의 자명할수 있다. 진행중에 있는 문제가 아니라 계단식세련에 집중할수 있도록 하기 위하여 이와 같은 잘 알려 진 실례가 심중히 선택된다.

> 월간 잡지 *True Life Software Disasters*에 대한 이름과 주소자료를 포함하고 있는 련속적인 주파일을 갱신하는 제품을 설계하시오. 여기에는 세가지 류형의 트랜잭션 즉 삽입, 수정, 삭제가 있는데 각각 트랜잭션 1, 2, 3이라고 한다. 이리하여 트랜잭션의 류형은 다음과 같다.
>
> 류형 1: INSERT(주파일에 대한 새로운 신청자)
> 류형 2: MODIFY(현존 신청자기록)

류형 3: DELETE(현존 신청자기록)

트랜잭션은 신청자의 이름에 대하여 자모순으로 정돈된다. 만일 주어 진 신청자에 대하여 하나이상의 트랜잭션이 존재하면 그 신청건에 대한 트랜잭션은 변경전에 삽입이 진행되고 삭제전에 변경이 진행되도록 정렬되였다.

풀이를 설계하는데서 첫 단계는 그림 5-1에서 보여 주는바와 같이 입력트랜잭션에 대한 하나의 전형적인 파일을 설치하는것이다.

트랜잭션의 형	이 름	주 소
3	Brown	
1	Harris	2 Oak Lane, Townsvill
2	Jones	Box 345, Tarrytown
3	Jones	
1	Smith	1304 Elm Avenue, Oak City

그림 5-1. 련속적인 주파일에 대한 입력트랜잭션기록

파일은 5개의 기록 즉 DELETE Brown, INSERT Harris, MODIFY Jones, DELETE Jones, INSERT Smith를 포함하고 있다(일단 동작과정에 같은 신청자에 대하여 변경과 삭제를 모두 진행할수 있다는것은 일반적이다.). 문제는 그림 5-2에서 보여 주는바와 같이 표현된다.

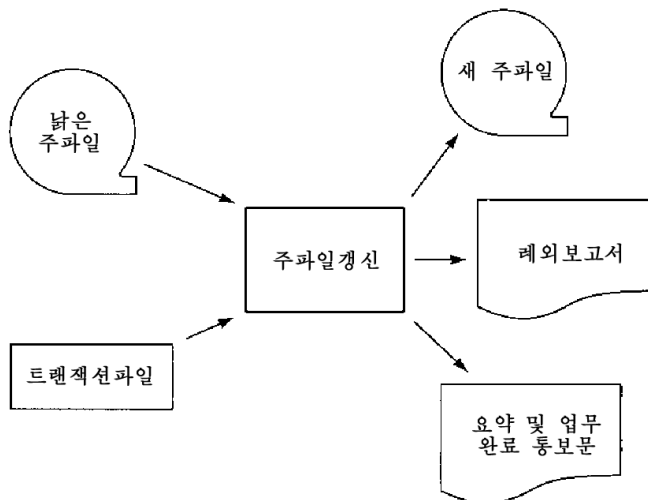



그림 5-2. 련속적인 주파일의 갱신

두개의 입력파일이 있다. 즉

1. 낡은 주파일이름과 주소기록

2. 트랜잭션파일

그리고 세개의 출력파일이 있다. 즉

3. 새로운 주파일이름과 주소기록
4. 례외보고서
5. 개괄과 업무완료통보문

설계과정을 시작하기 위하여 그림 5-3에서 보여 준바와 같이 첫 시작위치는 하나의 통 **주파일갱신**이다. 이 통은 **입력, 처리, 출력**이라고 하는 세개의 통으로 분해될수 있다. **처리**가 기록을 요구할 때 우리는 바로 정확한 시간에 정확한 기록들을 생성할수 있는 능력을 가지고 있다고 가정을 한다. 류사하게 우리는 정확한 시간에 정확한 기록을 정확한 파일에 쓸수 있다. 따라서 기술은 **입력**과 **출력**측면을 따로 분리하고 **처리**에 집중하는것이다. 이 **처리**는 무엇인가? 그것의 사명을 결정하기 위하여 그림 5-4에서 보여 준 실례를 고찰하자.

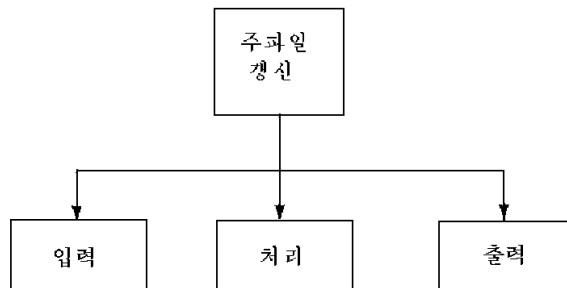



그림 5-3. 설계의 첫번째 세련

트랜잭션파일
3 Brown
1 Harris
2 Jones
3 Jones
1 Smith

낡은 주파일
Abel
Brown
James
Jones
Smith
Townsend

새 주파일
Abel
Harris
James
Smith
Townsend

례외보고서
Smith

그림 5-4. 트랜잭션파일, 낡은 주파일, 새 주파일 그리고 례외보고서

첫 트랜잭션기록(Brown)에 대한 열쇠를 첫 낡은 주파일기록(Abel)에 대한 열쇠와

비교한다. Brown이 Abel다음에 오기때문에 Abel기록은 새로운 주파일에 써주고 다음의 낡은 주파일기록(Brown)이 읽어 진다. 이 경우에 트랜잭션기록에 대한 열쇠는 낡은 주파일기록에 대한 열쇠와 정합되고 트랜잭션류형이 3(DELETE)이기때문에 Brown기록은 삭제되여야 한다. 이것은 Brown기록을 새로운 주파일에 복사시키지 않음으로써 실현된다. 다음 트랜잭션기록(Harris)과 낡은 주파일기록(James)이 읽어 지고 그것들의 개개의 완충기들에 Brown기록을 덧쓰기한다. Harris는 James앞에 있고 따라서 새로운 주파일에 삽입된다. 다음 트랜잭션기록(Jones)이 읽어 진다. Jones는 James다음에 있기때문에 James기록은 새로운 주파일에 씌여 지게 되고 다음 낡은 주파일기록이 읽어 진다. 즉 이것이 Jones이다. 트랜잭션파일에서 볼수 있는바와 같이 Jones기록은 변경되고 그다음 삭제되며 따라서 다음 트랜잭션기록(Smith)과 다음 낡은 주파일기록(역시 Smith)이 읽어 진다. 다행히도 트랜잭션류형이 1(INSERT)이지만 Smith는 이미 주파일에 있다. 따라서 자료에는 일정한 종류의 오유가 존재하며 Smith기록은 례외보고서에 씌여 진다. 더 정확히 보기 위하여 Smith트랜잭션기록은 례외보고서에 씌여 지고 Smith낡은 주파일기록은 새로운 주파일에 씌여 진다.

이제는 **처리**에 대하여 리해되기때문에 그것은 그림 5-5에서처럼 표현될수 있다.

트랜잭션기록건 = 낡은 기본 파일기록건	1. INSERT : 오유통보문을 인쇄한다. 2. MODIFY : 주파일기록을 변경한다. 3. DELETE : *주파일기록을 삭제한다.
트랜잭션기록건 > 낡은 기본 파일기록건	새 주파일에 낡은 주파일기록을 복사한다.
트랜잭션기록건 < 낡은 기본 파일기록건	1. INSERT : 새 주파일에 트랜잭션 기록을 써넣는다. 2. MODIFY : 오유통보문을 인쇄한다. 3. DELETE : 오유통보문을 인쇄한다.

* 주파일기록의 삭제는 새 주파일기록에 기록을 복사하지 않고 실현된다.

그림 5-5. 처리의 도식적인 표현

다음으로 그림 5-3의 **처리**통이 그림 5-6의 두번째 세련의 결과로 세련되게 된다.

입력과 **출력**통에 점선을 친것은 입력과 출력을 조종하는 방법에 대한 결정이 다음 세련까지 연기되였다는것을 의미한다. 그림에서 나머지부분은 **처리**에 대한 흐름도표이고 또는 오히려 앞선 흐름도표에 대한 초기의 세련으로 된다. 이미 강조한바와 같이 입력과 출력은 연기된다. 또한 파일의 끝조건에 대하여서는 대책이 없고 오유조건에 맞다들릴 때 무엇을 하여야 하는가 하는것이 아직 명시되지 않았다. 계단식세련의 우점은 이러한 문제들과 류사한 문제들이 이후 세련과정에 해결될수 있다는것이다.

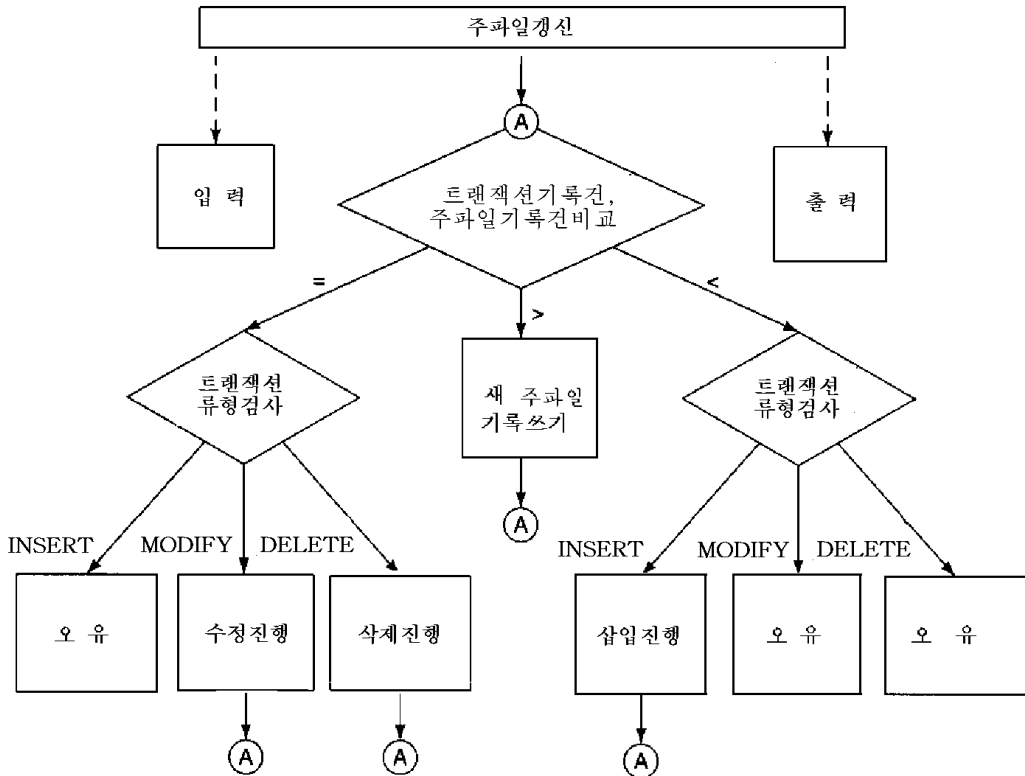



그림 5-6. 설계의 두번째 세련

다음단계는 그림 5-7을 초래하는 그림 5-6의 **입력**과 **출력**통을 세련시키는것이다. 파일끝조건의 끝이 여전히 조종되지 않았거나 또는 일감완료통보문이 아직 씌여 있지 않게 된다. 또한 이것들은 이후의 반복에서 수행될수 있다. 그러나 결정적인것은 그림 5-7의 설계가 한가지 기본오유를 가지고 있다는것이다. 이것을 보기 위하여 그림 5-4의 자료와 관련한 정황을 고찰하자. 이때 현재의 트랜잭션은 2Jones 즉 Jones가 변경되였으며 현재의 낡은 주파일기록은 Jones이다.

그림 5-7의 설계에서 트랜잭션기록에 대한 열쇠는 낡은 주파일기록에 대한 열쇠와 같기때문에 제일 왼쪽에 있는 통로는 **트랜잭션류형시험**결정통에로 따라 내려 간다. 현재의 트랜잭션류형이 MODIFY이기때문에 낡은 주파일기록은 변경되여 새로운 주파일에 씌여 지기때문에 다음의 트랜잭션기록이 읽어 진다. 이 기록은 3Jones이다. 즉 Jones를 삭제한다. 그러나 수정된 Jones기록은 이미 새로운 주파일에 씌여 졌다. 독자들은 무엇때문에 부정확한 세련이 의도적으로 제시되였는가고 이상하게 여길수도 있다. 요점은 계단식세련을 리용할 때 다음단계에로 넘어 가기전에 매개의 성공적인 세련에 대하여 탁상검열을 진행하는것이 필요하다는것이다. 만일 특정한 세련이 오유로 되여 버리면 그것은 처음부터 처리를 재시작할 필요는 없으나 이전의 세련으로 되돌아 가서 거기서부터 진행하여야 한다. 이 실례에서는 두번째 세련(그림 5-6)이 정확하다. 그것은 세번째 세련에서

다른 시도를 하기 위한 기초로서 리용될수 있다. 이번에는 설계는 준위1 전망(*lookahead*)을 리용한다. 즉 트랜잭션기록은 오직 다음의 트랜잭션기록이 분석된 다음에만 처리된다. 세부적인것들은 련습으로 남겨 둔다. 문제 5.1을 참고하시오.

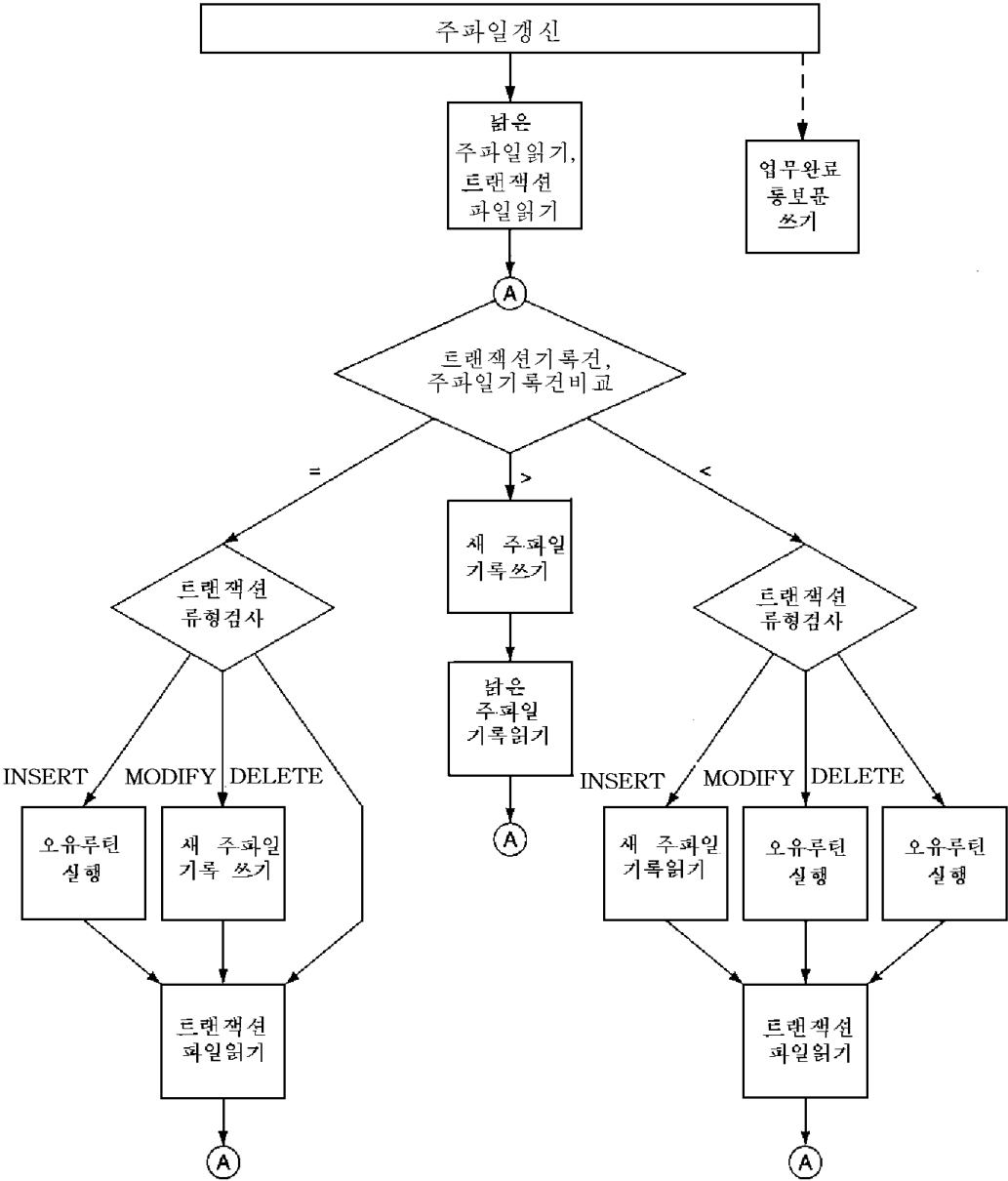



그림 5-7. 설계의 세번째 세련(설계에는 주요한 결함이 있다)

네번째 세련에서는 파일을 열고 닫는것과 같은 지금까지는 무시되여 온 세부들이 인입되여야 한다. 계단식세련에서 이와 같은 세부들은 마지막에 즉 론리적인 설계가 완전

히 개발된 다음에 조종된다. 명백하게 파일의 열기와 닫기를 진행하지 않고서는 제품을 실행할수 없다. 그러나 여기서 중요한것은 파일열기, 파일닫기와 같은 세부들이 조종되는 설계공정에서의 단계이다. 설계가 진행되고 있는 동안은 설계가들이 한번에 집중할수 있는 7개정도의 토막들은 파일열기와 닫기와 같은 세부들을 포함하지 말아야 한다. 즉 파일열기와 닫기는 설계 그자체에서는 아무것도 하지 말아야 하며 순전히 설계의 부분으로 되는 실현세부로 된다. 그러나 후의 세련에서는 파일열기와 닫기가 사활적인것으로 된다. 달리 말하면 계단식세련은 그 단계안에서 해결되여야 할 여러가지 문제들의 우선권을 설정하기 위한 하나의 기법으로 고찰될수 있다. 계단식세련은 임의의 한 순간에 7±2개 이상의 토막들을 조종하지 않고서도 매 문제가 풀릴수 있으며 그 매개는 적당한 시간에 풀린다는것을 담보한다.

용어 계단식세련(*stepwise refinement*)은 워스[Wirth, 1971]에 의하여 처음으로 도입되였다. 앞의 실례에서 계단식세련은 흐름도표에 적용되였다. 반면에 워스는 이 기법을 의사코드를 작성하는데 응용하였다. 계단식세련이 적용되는 특정한 표현은 중요치 않다. 즉 계단식세련은 쏘프트웨어개발의 매 단계에서 거의 모든 표현과 함께 리용될수 있는 일반적인 기법으로 된다.

계단식세련의 능력은 쏘프트웨어공학자들이 현재의 개발단계(앞의 실례에서 설계단계)에서의 관련 있는 측면들에 집중하도록 도와 주며 비록 전면적인 구조도식에서 본질적이라고 하더라도 고려할 필요가 없는 세부들을 무시하도록 도와 주는것이다. 사실상 이러한 세부들은 늦게까지 무시되여야 한다. 문제전체를 중요성에서 본질적으로 등가인 부분문제들로 분해하는 분할정복기법과는 달리 계단식세련에서는 문제의 특수한 측면이 가지는 중요성이 매 세련에서 변화되게 된다. 초기에는 어떤 특정한 문제가 련관이 없을수도 있지만 후에는 같은 론점들이 결정적으로 중요하게 될것이다. 계단식세련에서의 난관은 현재의 세련에서 어떤 문제점들은 조종되여야 하며 어떤 문제점들이 이후의 세련으로 미루어야 하는가를 결정하는것이다. 계단식세련과 같이 비용 대 리득분석은 쏘프트웨어생명주기전반에서 리용된 또 하나의 근본적이며 리론적인 쏘프트웨어공학기법이다. 이 기법은 다음절에서 설명한다.

5. 2. 비용 대 리득분석

어떤 가능한 작용과정이 유익한가를 결정하는 하나의 방법은 계획된 장래비용과 타산된 장래리득을 비교하는것이다. 이것을 비용 대 리득분석(*cost-benefit analysis*)이라고 한다. 콤퓨터와 관련한 범위에서 비용 대 리득분석의 실례로 1965년에 크래그중앙전기회사(KCEC; *Krag Central Electric Company*)가 계산서작성체계의 콤퓨터화실현문제를 어떻게 결정하였는가를 고찰해 보자. 계산서작성은 KCEC손님들에게 두달에 한번씩 계산서를 우편발송해 주는 80여명의 사무원들에 의해서 수동적으로 진행되고 있었다. 콤퓨터화를 실현하기 위하여 KCEC는 착공카드나 자기테프에 입력자료를 기록하기 위한 자료획득설비를 비롯하여 필요한 쏘프트웨어와 하드웨어를 사거나 임대하여야 했다.

콤퓨터화의 한가지 우월성은 계산서가 두달에 한번이 아니라 매달 발송될수 있으며

따라서 회사의 현금수입이 상당히 개선될수 있다는것이다. 더우기 80여명의 계산서작성사무원들이 11명의 자료작성사무원들로 교체되게 된다. 로임절약은 157만 5천딸라로 평가되였고 개선된 현금수입은 87만 5천딸라의 가치를 가지게 되였다. 따라서 전체적인 리득은 245만딸라로 평가되였다. 다른 한편 높은 로임을 받는 콤퓨터전문가들이 지도하는 완전한 자료처리부서를 설치하여야 하였다. 7년이 지나서 비용은 다음과 같이 평가되였다. 유지정비를 포함하여 쏘프트웨어와 하드웨어의 가격은 125만딸라로 평가되였다. 첫해에는 35만딸라의 전환비용이 들었고 새 체계를 손님들에게 설명되는데 든 비용은 추가적으로 12만5천딸라로 평가되였다. 전체 가격은 172만 5천딸라로 평가되였고 평가된 리득보다도 약 75만딸라가 더 적었다. KCEC는 즉시에 콤퓨터화를 진행하기로 결정하였다.

비용 대 리득분석은 언제나 늘 단순한것은 아니다. 한편 어떤 관리고문은 로임절약을 타산할수 있으며 부기원도 현금수입에서의 개선을 계획할수 있다. 그리고 순리득금[Yourdon, 1989]은 화페가치의 변경을 조종하는데 리용될수 있으며 쏘프트웨어공학고문은 하드웨어와 쏘프트웨어의 가격과 그리고 그것들로 전환시키는데 드는 가격을 타산할수 있다. 그러나 콤퓨터화에 적응하기 위하여 노력하는 손님들을 위하여 지출되는 비용을 어떻게 결정하겠는가? 또는 홍역에 대처하여 전체 주민들을 접종시키는것으로부터 얻는 리득은 어떻게 측정하겠는가?

중요한것은 명백한 리득은 측정하기가 쉽지만 막연한 리득은 적접 정량화하기가 힘들다는것이다. 막연한 리득에 대하여 딸라가격을 부여하는 현실적인 방도는 가설(*assumption*)을 세우는것이다. 이러한 가설들은 언제나 결과적인 리득타산과 결합되여 언급되여야 한다. 결국 관리자들은 결심을 해야 한다. 만일 리용가능한 자료가 없다면 이러한 자료로부터 일반적으로 결정되는 가설확립은 환경에 기초하여 수행될수 있는 가설이 가장 좋은 방도로 된다. 이러한 방법은 만일 자료와 기초를 이루고 있는가를 검토하는 누군가가 더 좋은 가설을 제기할수 있다면 더 좋은 자료가 생성되고 그와 관련한 막연한 리득이 보다 더 정확하게 계산될수 있다는 우점이 있다. 그러한 기법은 막연한 가격을 결정하는데 리용될수 있다.

비용 대 리득분석은 손님들이 자기의 업무를 콤퓨터화하는가, 그렇게 한다면 어떤 방법으로 하겠는가를 결정하는데서 중요한 기법으로 되고 있다. 여러가지 변화된 전략들에 대하여 가격과 리득이 비교된다. 실례로 약품시험결과를 보관하는 제품은 보통의 파일과 여러가지 자료기지관리체계를 비롯하여 여러가지 방법으로 실현될수 있다. 매 가능한 전략에 대하여 가격과 리득을 계산하고 가격과 리득차가 최대인것들중에서 하나를 최량인 전략으로 선택한다.

이 장에서 서술한 마지막 리론적인 도구는 쏘프트웨어척도이다.

5. 3. 쏘프트웨어의 척도

2.11에서 서술한바와 같이 측정(또는 척도)이 없으면 문제점들을 걷잡을수 없게 쏘프트웨어개발공정에서 신속하게 이러한 문제점들을 발견해 낼수 없다. 이리하여 척도는 가능한 잠재적인 문제들에 대하여 신속하게 경고를 주는 체계로써 쓰일수 있다. 실례로 코

드의 행들은 제품의 크기를 측정하는 방도의 하나로 된다(9.2.1). 만일 LOC측정이 규칙적인 간격으로 진행되면 그것은 프로젝트개발이 얼마나 빨리 진행되고 있는가를 측정할수 있게 해준다. 이밖에 코드 1,000행당 결함(오유)의 수는 쏘프트웨어의 품질에 대한 측정으로 된다. 결국 프로그람작성자가 한달동안에 2,000행이 되는 코드를 모순이 없게 작성하였지만 그 절반은 접수할수 없기때문에 버려야 한다면 조금밖에 리용되지 않게 된다. 따라서 LOC 하나로서는 그리 중요한 척도로 되지 않는다.

일단 제품이 의뢰자의 콤퓨터에 설치되면 오유들사이의 평균시간과 같은 척도는 관리자측에 그 믿음성에 대하여 지적해 준다. 만일 어떤 제품이 매일 실패하면 그 품질은 실패가 없이 평균적으로 9개월동안 동작한 류사한 제품의 품질보다 더 낮다.

정확한 척도는 쏘프트웨어개발공정전반에 걸쳐 리용될수 있다. 실례로 매 단계에 대하여 한사람당 한달동안에 들인 노력을 측정할수 있다. 직원이동비률은 또 하나의 중요한 척도로 된다. 새로 온 종업원은 프로젝트와 관련한 일들에 익숙하는데 시간이 들기때문에 현재의 프로젝트에 불리한 영향을 준다(4.1). 이밖에 새로 온 종업원은 쏘프트웨어개발공정의 여러 측면에서 교육을 받아야 할수 있다. 즉 만일 새로 온 종업원이 교체된 개별적인 사람들보다도 쏘프트웨어공학에 대해서 더 적게 교육을 받았다면 그때는 개발공정전체가 애로를 느끼게 된다. 물론 가격은 역시 개발공정전반에 대하여 계속 관리되여야 할 필수적인 척도이다.

이 책에서는 여러가지 서로 다른 척도들이 서술된다. 일부 척도들은 제품의 척도(*product metrics*)이다. 즉 그것들은 제품의 크기라든가 제품의 믿음성과 같은 제품 그자체에 관한 일부 측면을 측정한다. 대비적으로 다른 척도들은 공정척도(*process metrics*)이다. 즉 이러한 척도들은 개발자들로 하여금 쏘프트웨어개발공정에 대한 정보를 도출하도록 하는데 리용된다. 이런 종류의 전형적인 척도는 개발기간에서의 오유발견효률 즉 제품이 개발리용되는 전 기간에 걸쳐 제품에서 발견되는 전체 오유개수에 대한 개발기간에 발견된 오유개수의 비률이다.

많은 척도들은 주어 진 단계에 특정한것들이다. 실례로 코드의 행들은 실현이 시작되기전에는 리용될수 없다. 그리고 명세서를 검토하는데서 시간당 발견된 오유개수는 다만 명세작성단계와만 관련된다. 쏘프트웨어개발공정의 여러 단계들을 서술한 다음의 장들에서는 그 장들과 관련된 척도들을 론의하고 있다.

비용은 척도값을 계산하는데 필요한 자료를 수집하는데 포함된다. 지어 자료수집이 완전히 자동화되면 요구되는 정보를 축적하는 CASE도구(5.4)가 무료가 아니며 그 도구로부터 나온 결과들을 해석하는데 노력이 든다. 수백개(수천개는 아니다.)의 서로 다른 척도가 앞으로 제기된다는것을 념두에 두면 한가지 명백한 문제는 쏘프트웨어기업체는 무엇을 측정하여야 하는가 하는것이다. 다음과 같은 5개의 본질적이며 근본적인 척도가 있다.

1. 크기(코드행수의 크기, 더 좋기는 9.2.1에서 언급한것과 같은 보다 의미 있는 척도에서의 크기로)
2. 가격(딸라로)
3. 기간(달수로)
4. 효과(한사람당 한달동안의 작업량으로)

5. 품질(발견된 오유의 개수로)

매 척도들은 단계마다 측정되여야 한다. 이러한 근본적인 척도로부터 나온 자료에 기초하여 관리자측은 설계단계에서의 높은 오유비률 또는 산업적기준이하의 코드출력과 같은 쏘프트웨어기업체내에서 제기되는 문제들을 식별할수 있다. 일단 문제령역이 강조되면 이러한 문제점들을 정정하기 위한 전략이 고찰되여야 한다. 이러한 전략을 성공적으로 조종하기 위하여 보다 더 세분화된 척도가 도입될수 있다. 실례로 매 프로그람작성자들의 오유비률에 대한 자료를 수집하거나 사용자들의 만족성정도를 측정하도록 하는것이다. 이처럼 5개의 기본적인 척도외에 보다 더 상세한 자료종합과 분석은 어떤 특정한 목적을 위하여 수행되여야 한다.

마지막으로 척도의 한 측면은 아직 더 론의할 여지가 있다. 대다수의 대중적인 척도의 타당성에 대하여 질문이 제기되였다. 즉 이러한 일부 문제들은 14.8.2에서 론의를 진행한다. 우리가 쏘프트웨어개발공정을 측정할수 없는 한 그것을 조종할수 없다는것은 사실이라 할지라도 무엇이 측정되여야 하는가에 관해서는 일정한 불일치가 존재한다 [Fenton and Pfleeger, 1997].

이제 리론적인 도구들로부터 콤퓨터지원쏘프트웨어공학(CASE)도구로 돌아 가 보자.

5. 4. 콤퓨터지원쏘프트웨어공학(CASE)

쏘프트웨어제품의 개발기간에는 서로 다른 여러가지 조작들이 수행되여야 한다. 전형적인 활동은 자원요구사항들의 평가, 명세서의 작성, 통합시험의 수행, 사용자지도서의 작성을 포함하게 된다. 그러나 유감스럽게도 이런 활동과 쏘프트웨어개발공정에서의 그밖의것들은 인간의 개입이 없이는 콤퓨터에서 완전히 자동화되거나 실행될수 없다.

그러나 콤퓨터가 매 단계를 방조할수는 없다. 이 절의 제목인 CASE는 콤퓨터지원(또는 콤퓨터방조)쏘프트웨어공학을 의미한다. 콤퓨터는 계획과 계약, 명세서, 설계, 원천코드와 관리정보와 같은 모든 종류의 인공적인 제품의 생성과 구성을 비롯하여 쏘프트웨어개발과 관련된 많은 어려운 작업의 수행을 도울수 있다. 문서작성은 쏘프트웨어개발과 유지정비에서 필수적이지만 쏘프트웨어개발에 종사하는 대다수의 사람들은 문서를 새로 작성하거나 갱신하는것을 좋아 하지 않는다. 콤퓨터들에 대한 유지정비선도는 쉽게 변경될수 있기때문에 특히 쓸모가 있다.

그러나 CASE는 문서작성을 지원하는데 국한되지 않는다. 특히 콤퓨터는 쏘프트웨어공학자들로 하여금 쏘프트웨어개발의 복잡성에 대처하도록 세부적인것들을 관리하는데 도움을 줄수 있다. 그것은 쏘프트웨어공학을 위한 콤퓨터지원의 모든 측면을 포함한다. 동시에 CASE는 콤퓨터자동프로그람공학이 아니라 콤퓨터지원프로그람공학을 의미한다는것을 기억하는것이 중요하다. 즉 콤퓨터는 아직 쏘프트웨어의 개발이나 유지정비와 관련하여 사람을 대신할수 없다는것이다. 앞으로도 콤퓨터는 기껏해야 쏘프트웨어전문가들의 도구로 될것이다.

5. 5. 콤퓨터지원쏘프트웨어공학(CASE)의 분류

CASE의 가장 간단한 형식은 쏘프트웨어도구(*tool*) 즉 쏘프트웨어생산의 한 측면만 지원하는 제품이다. CASE도구는 현재 생명주기의 모든 단계에서 리용된다. 실례로 다양한 여러가지 도구들이 나오고 있는데 대부분의 도구들은 흐름선도와 같은 쏘프트웨어제품의 도형적표현들을 구성하는데 도움을 주는 개인용콤퓨터에서 리용된다. 개발공정의 보다 초기단계(즉 요구사항확정과 명세작성 그리고 설계단계)에서 개발자들을 방조하는 CASE도구들은 흔히 상위CASE(*upperCASE*) 또는 앞단(*front-end*)도구라고 부르며 실현단계와 통합단계, 유지정비단계를 지원하는 CASE도구들은 하위CASE(*lowerCASE*) 또는 뒤단(*back-end*)도구라고 부른다. 그림 5-8의 ㄱ)는 요구사항확정단계의 부분들을 지원하는 CASE도구를 보여 주고 있다.

ㄱ) 도구	ㄴ) 작업대	ㄷ) 환경
요구사항확정단계	요구사항확정단계	요구사항확정단계
명세작성단계	명세작성단계	명세작성단계
설계단계	설계단계	설계단계
실현단계	실현단계	실현단계
통합단계	통합단계	통합단계
유지정비단계	유지정비단계	유지정비단계

그림 5-8. 도구, 작업대, 환경

CASE도구에서 중요한 도구의 하나는 제품에서 정의된 모든 자료들의 콤퓨터화된 목록인 자료사전(*data dictionary*)이다. 큰 제품들은 수만개(수십만개는 아니다.) 자료항목을 포함하게 되며 콤퓨터는 변수의 이름과 행, 그것이 정의되여 있는 위치 그리고 처리절차의 이름과 파라메터, 파라메터의 형과 같은 정보들을 보관하는데서 리상적이다. 매 자료사전입력점에 대하여 가장 중요한 부분은 항목들을 서술하는것이다. 실례로 **새로 출생한 갓난 애기의 몸무게를 입력으로 하고 한번에 먹여야 할 약의 량을 계산하는 처리절차라든가 또는 비행기도착시간을 가장 빠른 시간순서로 정돈**하여 놓은 목록과 같은 항목들을 서술하고 있다.

자료사전의 능력은 일관성검사기(*consistency checker*)와 결합되여 강화될수 있다. 즉

명세서에 있는 매 자료항목이 설계단계에 반영되여 있고 반대로 설계단계에 있는 매 항목이 명세서에서 정의되였는가를 검사할수 있는 도구와 결합됨으로써 강화될수 있다. 자료사전의 또 다른 리용은 보고서생성프로그람(*report generator*)과 화면생성프로그람(*screen generator*)에 대한 자료를 제공하는것이다. 보고서생성프로그람은 보고서를 만드는데 필요한 코드를 생성하는데 리용된다. 화면생성프로그람은 자료를 현시하는 화면을 구성하기 위한 코드를 작성하는데서 쏘프트웨어개발자들을 지원한다. 련쇄된 책방들의 매 가지에서 주당 판매를 입력하기 위한 하나의 화면을 설계한다고 하자. 가지의 수는 1,000~4,500까지 또는 8,000~8,999까지의 범위에 있는 4자리옹근수이며 화면에서 꼭대기에 있는 세개의 행에 입력된다. 이 정보는 화면생성프로그람에 보내여 진다. 화면생성프로그람은 그다음 자동적으로 꼭대기부터 세개행에 문자렬 BRANCH NUMBER----을 현시하고 밑줄을 그은 첫 문자들에 유표를 표시하도록 하는 코드를 생성한다. 사용자가 매 수자를 입력하는데 따라 그것이 현시되고 유표는 다음 밑줄 그은 위치로 이동한다. 화면생성프로그람은 사용자가 수자만을 입력했는가 결과적인 4자리옹근수가 특정한 범위에 있는가를 검사하고 코드를 생성한다. 만일 자료가 타당치 못하게 입력되거나 사용자가 ?건을 누르면 도움말정보가 현시된다.

이러한 프로그람의 리용은 재빨리 구축되는 신속원형에 귀착될수 있다. 더우기 자료사전과 일관성검사프로그람, 보고서생성프로그람과 화면생성프로그람을 모두 결합한 도형적인 표현도구는 신속원형작성을 지원하는 명세작성과 설계작업대(*workbench*)를 구성한다. 이 모든 특성을 병합한 작업대의 실례*)로는 PowerBuilder, Software through Pictures과 System Architect를 들수 있다.

CASE작업대는 이리하여 하나 또는 두가지 활동을 함께 지원하는 도구들의 집합으로 되는데 여기서 활동은 련관된 과제들의 집합으로 된다. 실례로 코드작성은 편집, 콤파일, 련결, 시험, 오유검사를 포함하고 있다. 활동은 생명주기모형에서의 단계와 같은것이 아니다. 사실상 어떤 활동에서의 과제들은 지어 단계들의 경계를 교차할수도 있다. 실례로 프로젝트관리작업대는 프로젝트의 매 단계에 리용되며 코드작성작업대는 실현과 통합, 유지단계는 물론 신속원형작성에 리용될수 있다. 그림 5-8의 ㄴ)은 상위 CASE도구들에 대한 작업대를 보여 주고 있다. 이 작업대는 명세작성과 설계단계의 부분들을 위한 도구는 물론 그림 5-8의 ㄱ)의 요구사항확정단계의 도구를 포함한다.

CASE기술은 도구로부터 작업대로 계속 발전하고 있기때문에 다음사항은 CASE환경(*environment*)이다. 하나 또는 두개의 활동을 지원하는 작업대와는 달리 환경은 완전한 쏘프트웨어개발공정 또는 적어도 쏘프트웨어개발공정의 큰부분을 지원한다[Fuggetta, 1993]. 그림 5-8의 ㄷ)은 생명주기의 모든 단계의 모든 측면을 지원하는 환경을 보여 주고 있다. 환경들에 대해서는 제14장에서 보다 더 상세히 론의한다.

CASE의 분류(즉 도구, 작업대 그리고 환경)를 설정하였으므로 다음으로 CASE의 범

---

*) 특정한 CASE도구를 이 책에서 인용한것은 저자나 출판업체들에 의한 그 CASE도구의 보증에 대한 어떤 형식을 암시하고 있다. 이 책에서 언급한 매 CASE도구는 실례로 되는 CASE도구들에서 전형적인 실례이기때문에 포함시켜 주었다.

위에 대하여 고찰한다.

5. 6. 콤퓨터지원쏘프트웨어공학(CASE)의 범위

이미 언급한바와 같이 언제나 리용할수 있는 정확하고 현대적인 문서는 CASE기술을 리용하는데서 나서는 1차적인 요구로 되고 있다. 실례로 명세서가 수동적으로 작성된다고 하자. 개발팀성원들은 특정한 명세서가 현재의 판본인가 낡은 판본인가를 말할 방도가 없다. 만일 그 문서상에서 손으로 써서 변경시킨것이 현재의 문서인지 또는 후에 제기된것인지 알 방도가 없다. 다른 한편 만일 제품에 대한 명세서들이 CASE도구를 리용하여 만들어 지면 그때는 CASE도구를 리용하여 접근할수 있는 직결식판본인 명세서에 대한 하나의 복제본만이 존재한다. 그때 만일 명세서가 변화되면 개발팀성원들은 쉽게 문서에 접근할수 있으며 그것들이 현재의 판본으로 보인다고 믿는다. 이밖에 일관성검사기는 명세서에 대응하는 변경을 진행하지 않고 임의의 설계변경들을 기발로 표시할것이다.

프로그람작성자는 또한 직결식문서를 요구한다. 실례로 직결식방조정보는 조작체계, 편집기, 프로그람작성언어에 대하여 제공되여야 한다. 이밖에 프로그람작성자는 편집지도서, 프로그람작성지도서와 같은 많은 종류의 지도서들을 참고하여야 한다. 가능한 모든 곳에서 이러한 지도서들을 직결로 리용할수 있게 하는것이 아주 바람직하다. 모든것을 가까이에 놓고 있는 편리성과는 달리 보통 적합한 지도서를 찾아 내고 필요한 항목을 찾아 보는것보다는 콤퓨터에 질문하는것이 더 빠르다. 이밖에 보통 회사안에서 지도서에 대한 모든 하드복사를 찾아 내고 필요한 페지들을 변경하려고 시도하기보다는 하나의 직결식지도서를 갱신하는것이 훨씬 더 쉽다. 결과 직결식문서는 같은 자료에 대한 하드복사보다 더 정확한것 같다. 이것이 직결식문서를 프로그람작성자에게 제공해 주게 되는 또 하나의 리유로 된다. 이와 같은 직결식문서의 실례는 UNIX지도서이다[Sobell, 1995].

CASE는 또한 팀성원들사이의 통신을 방조할수 있다. 전자우편은 콤퓨터나 확스와 같은 보통 사무처리를 훨씬 고속으로 해주고 있다. 전자우편은 우점이 많다. 쏘프트웨어 제품의 견지에서 보면 특정한 프로젝트와 관련한 모든 전자우편에 대한 복사들이 특정한 우편통에 저장되면 거기에는 프로젝트를 개발하는 기간에 만들어 지게 되는 결정들을 저장한 기록이 있게 된다. 이것은 후에 일어 날수 있는 충돌을 해소하는데 리용된다. 많은 CASE환경들과 일부 CASE작업대는 지금 전자우편체계를 병합하고 있다. 기타 회사들에서는 전자우편체계를 Netscape와 같은 WWW열람기를 통하여 실현하고 있다. 이와 맞먹는 도구들가운데는 표처리프로그람과 문서처리프로그람이 있다. 용어코드작성도구(*coding tool*)는 본문편집기, 오유수정프로그람 그리고 프로그람작성자의 과제를 단순하게 하고 많은 프로그람작성자의 능률을 높이기 위하여 설계된 기묘한 인쇄프로그람(*pretty printer*)과 같은 CASE도구들과 련관된다. 이러한 도구들을 론의하기전에 다음의 세가지 정의가 필요하다. 소규모프로그람작성(*programming-in-the-small*)은 단일모듈코드준위에서의 쏘프트웨어개발로 간주되며 반면에 대규모프로그람작성(*programming-in-the-large*)은 모듈준위의 쏘프트웨어개발이다[DeRemer and Kron, 1976]. 후자는 구성방식설계와 통합과 같은 측면을 포함하게 된다. 다수프로그람작성(*programming-in-the-many*)은

대규모프로그람작성과 소규모프로그람작성의 두 측면들을 결합하고 있다.

구조편집기(*structure editor*)는 실현언어를 《리해》하는 본문편집기이다. 즉 구조편집기는 무익한 콤파일에서 시간이 랑비되기때문에 실현단계를 앞당기기 위하여 프로그람작성자가 건반으로 입력하자마자 문법적인 오유를 찾아 낼수 있도록 한다. 구조편집기는 매우 다양한 언어들과 조작체계, 하드웨어내에 포함되여 있다. 구조편집기는 프로그람작성언어에 대한 지식을 가지고 있기때문에 코드의 좋은 시각적출현을 늘 담보할수 있도록 인쇄프로그람(또는 형식화프로그람)을 본문편집기에 쉽게 병합할수 있다. 실례로 C++를 위한 기묘한 인쇄프로그람은 그에 대응하는 }가 대응하는 {만큼 같이 들여쓰기하도록 담보해 준다. 형식화프로그람을 병합한 구조편집기의 초기의 실례는 마킨토쉬, 파스칼편집기이다[Apple, 1984]. 예약된 단어들 두드러 지게 나타나도록 자동적으로 강조체로 되며 들여쓰기는 읽기 쉽게 하기 위하여 주의 깊게 설계되였다. 사실상 많은 마킨토쉬본문편집기들은 전체적으로 또는 부분적으로 구조화되여 있다.

다음으로 어떤 방법이 존재하지 않거나 그것이 어떤 방식으로 잘못 규정되였다는것을 련결과정에 발견하기 위하여 코드내에서 그 방법을 호출하는 문제를 생각해 보자. 이때 구조편집기가 직결식대면부검사(*online interface checking*)를 지원하도록 하는것이 필요하다. 즉 구조편집기가 프로그람작성자에 의해서 선언된 매 변수의 이름과 관련한 정보를 가질 때와 마찬가지로 제품안에서 정의된 매 방법들의 이름들도 알아야 한다. 실례로 프로그람작성자가 다음과 같은 호출

average = dataArray.computeAverage(numberOfValues);

을 입력하였지만 방법 computeAverage가 아직 정의되지 않았다면 그때 편집기는 즉시에 다음의 통보문

Method computeAverage not known

으로 응답한다.

이 점에서 프로그람작성자는 방법의 이름을 정정하든가 또는 computeAverage라는 새로운 방법을 정의하든가 하는 두가지 선택을 하게 된다. 만일 두번째를 선택한다면 프로그람작성자는 또한 새로운 방법에 대한 인수들을 목록에 기입하여야 한다. 직결식대면부검사를 하는 주요한 원인이 바로 방법의 이름이 아니라 전체의 대면부정보를 정확히 검열할수 있게 하는것이기때문에 새로운 방법을 정의할 때 인수의 형을 주어야 한다. 일반적인 오유는 방법 q가 5개의 인수를 가지는데 방법 p가 이를테면 4개의 인수를 넘겨 주는 방법 q를 호출하는것이다. 호출이 정확히 4개의 인수를 리용하고 있지만 두개의 인수가 바뀌여 져 있을 때는 오유를 발견하기가 더욱 어렵다. 실례로 방법 q의 선언이

void q(**float** floatVar, **int** intVar, string s1, string s2)

이고 반면에 호출은

q(intVar, floatVar, s1, s2);

이다. 호출명령문에서 첫 두개의 인수는 바뀌여 져 있다. Java콤파일러와 련결프로그람은

이 오유를 발견은 하지만 다만 그것들이 그후에 호출될 때에만 그렇게 될수 있다. 대비적으로 직결식대면부검사기는 즉시 이러한 오유와 이와 류사한 오유들을 발견한다. 이밖에 편집기가 방조능력을 가진다면 프로그람작성자는 q에 대한 호출을 코드작성하기전에 방법 q의 정확한 인수에 대한 직결정보를 요구할수 있다. 지금은 더 좋게 되여 있다. 편집기는 호출에 대한 형타를 생성하여 매 인수에 대한 형을 보여 주고 있다. 프로그람작성자는 단지 정확한 형에 따르는 실제적인 인수를 매 형식적인 인수로 교체하여야 한다.

직결식대면부검사의 주요한 우점은 인수의 개수가 맞지 않게 방법을 호출한다거나 또는 형이 맞지 않게 방법을 호출함으로써 일어 나는 발견하기 힘든 오유들을 즉시에 표식해 주고 중지시킨다는것이다. 직결식대면부정보는 특히 쏘프트웨어가 팀에 의해서 개발될 때(다수프로그람작성) 품질이 좋은 쏘프트웨어를 효률적으로 생산하는데서 중요한 역할을 한다. 모든 모듈들과 관련한 직결식대면부정보는 전 기간 모든 프로그람작성팀성원들에게 리용될수 있다는것이 본질적인 문제이다. 더우기 한명의 프로그람작성자가 한개 인수의 형을 **int**에서 **float**로 변화시키거나 또는 추가적인 인수를 보충함으로써 방법 vaporCheck의 대면부를 변경시키면 vaporCheck를 호출하는 모든 구성부분들은 련관된 호출명령문들이 사건의 새로운 상태를 반영하기 위하여 바뀌여 질 때까지 자동적으로 무시되여야 한다.

지어 직결대면부검사기를 병합한 문법지향편집기를 가지고서도 프로그람작성자는 여전히 편집기로부터 탈퇴하여 콤파일러와 련결프로그람을 호출해야 한다. 명백히 콤파일오유는 없지만 콤파일러는 여전히 코드생성을 하도록 호출되여야 한다. 그다음 련결프로그람이 호출되여야 한다. 또한 프로그람작성자는 직결대면부검사기가 존재하는것으로 하여 모든 외부적인 참고가 만족되지만 련결프로그람은 아직도 제품을 련결하기 위하여 필요된다고 믿을수 있다. 이에 대한 해결책은 편집기내에서 조작체계앞단처리(*operating system front-end*)를 병합하는것이다. 즉 프로그람작성자는 조작체계에 편집기로부터 오는 명령들을 줄수 있어야 한다. 편집기가 콤파일러와 련결프로그람, 적재프로그람과 그리고 모듈이 실행되도록 하여야 할 필요가 있는 기타 다른 체계쏘프트웨어를 호출하도록 하기 위하여 프로그람작성자는 GO 또는 RUN이라는 간단한 지령을 입력하거나 마우스로 적당한 그림기호나 차림표를 선택할수 있다. UNIX에서는 이것을 *make*지령(5.9)을 리용하거나 쉘스크립트(*shell script*)를 호출하여 실현시킬수 있다[Sobell, 1995]. 이러한 앞단처리는 다른 조작체계들에서도 역시 실현하고 있다.

가장 실망케 하는 계산경험은 제품으로 하여금 1s동안에 실행하게 하고 그다음에는

Overflow at 506

과 같은 통보문을 인쇄하면서 다급히 끝내는것이다. 프로그람작성자는 아셈블리나 기계코드와 같은 낮은 준위의 언어가 아니라 Java나 C++와 같은 고급언어를 가지고 프로그람작성을 한다. 그러나 오유수정기능지원이 Overflow at 506의 변종들일 때 프로그람작성자는 기계코드핵담프, 아셈블리어목록작성, 련결프로그람목록작성, 기타 여러가지 낮은 준위언어문서들을 시험해야 하며 그로 말미암아 고급언어프로그람작성의 총체적인 우월성이 상실되게 된다. 주어 진 유일한 정보가 잘 알려 지지 않은 UNIX통보문

Core dumped

이거나 마찬가지로 유익하지 못한

Segmentation fault

인 경우에 이와 류사한 정황이 발생한다. 여기서 다시 사용자는 낮은 준위의 정보를 시험하여야 한다.

실패와 관련하여 그림 5-9에서 보여 준 통보문은 앞에서의 간결한 오유통보문에 비해서 큰 개선으로 된다. 프로그람작성자는 즉시에 0으로 나누기하여 방법이 실패하였다는것을 알수 있다. 조작체계에 대하여 편집방식을 입력하고 오유가 발견된 행 즉 례하면 6행을 그 앞뒤의 4~5개행과 함께 자동적으로 현시하도록 하는것이 더욱 쓸모 있다. 그러면 프로그람작성자는 실패한것이 무엇이고 어떤 변경을 진행해야 하는가를 알수 있다. 또 다른 류형의 원천준위의 오유검사는 추적이다. CASE도구들이 출현하기전에는 프로그람작성자들이 코드에 수동적으로 적당한 입력지령을 삽입하여야 하였다. 즉 실행할 때 행번호와 관련된 변수들의 값을 가리키도록 하였다. 지금은 이것을 자동적으로 측정결과를 생성해 내도록 하는 원천준위오유수정프로그람(*source-level-debugger*)에 지령을 주어 실현하고 있다. 더 좋은 대화식원천준위 오유수정프로그람(*interactive source-level-debugger*)이 있다. 변수 escapeVelocity 의 값이 정확치 않고 방법 computeTrajectory에 오유가 있는것 같다고 가정하자. 대화식원천준위 오유수정프로그람을 리용하여 프로그람작성자는 코드상에 정지점을 설정할수 있다. 정지점에 이르게 되면 실행이 정지되고 오유수정방식으로 들어 간다. 프로그람작성자는 이제 오유수정프로그람에 변수 escapeVelocity와 방법 computeTrajectory를 추적하라고 요구한다.

OVERFLOW ERROR

```
Class:      cyclotronEnergy
Method:     performComputation
Line 6:     newValue = (oldValue + tempValue)/tempValue;
                    oldValue = 3.9583        tempValue = 0.000
```

그림 5-9. 원천준위오유수정프로그람의 출력

즉 escapeVelocity의 값이 련속적으로 리용되기도 하고 변화되기도 할 때마다 매번 실행이 다시 중지된다. 그때 프로그람작성자는 또 오유수정지령을 입력하게 된다. 실례로 특정한 변수의 값이 현시되도록 요구하게 한다. 프로그람작성자는 오유수정방식으로 계속 실행하는가 또는 보통실행방식으로 되돌아 가는가를 번갈아 선택할수 있다. 프로그람작성자는 방법 computeTrajectory가 입력되거나 중지될 때마다 오유수정프로그람과 대화할수 있다. 이런 대화식원천준위오유수정프로그람은 제품이 실패할 때 프로그람작성자에게 모든 형태의 도움을 주게 된다. UNIX오유수정프로그람 dbx는 이와 같은 CASE도구의 실례로 된다.

이미 강조한바와 같이 모든 종류의 문서가 직결로 리용될수 있다는것이 중요하다. 프로그람작성자의 경우에 필요한 모든 문서는 편집기안에서부터 접근가능하여야 한다. 방금 서술한것들 즉 직결식대면부검사능력, 조작체계앞단처리, 원천준위오유수정프로그람, 직결식문서작성들을 가지고 있는 구조편집기는 충분하고도 효과적인 작업대를 구성한다.

이런 종류의 작업대는 결코 새로운것이 아니다. 앞에서 렬거한 모든 특성은 1980년이전의 FLOW쏘프트웨어개발작업대에서 지원되였다[Dooley and Schach, 1985]. 그러므로 최소이지만 본질적인 프로그람작성작업대로서 제시될수 있게 되기전에는 여러해동안의 연구를 필요로 하지 않는다. 아주 상반되게 필요한 기술은 20년이상 걸렸고 싼(Sun)쏘프트웨어회사와 같이 작업대를 리용할 대신에 여전히 《낡은 방식》으로 코드작성을 실현하는 프로그람작성자들도 있다는것은 놀라운 일이다.

특히 팀이 쏘프트웨어를 개발할 때 필수적인 도구는 판본조종도구이다.

5. 7. 쏘프트웨어판본

제품이 유지정비될 때마다 적어도 제품에 대하여 두개의 판본 즉 낡은 판본과 새로운 판본이 있게 된다. 제품은 모듈들로 구성되기때문에 역시 변화가 진행된 매 구성모듈에 대하여 두개이상의 판본이 있게 된다.

판본조종은 처음에는 유지정비단계의 범위안에서 서술되여 있고 그다음에는 개발공정의 보다 앞단계들을 포함하도록 확장되였다.

5. 7. 1. 개정판

어떤 제품이 여러 곳에 설치되였다고 하자. 만일 모듈에서 오유가 발견되면 그 모듈은 보수되여야 한다. 적당히 변경을 진행한 다음에는 그 모듈에 대하여 두개의 판본이 존재하게 된다. 즉 교체하려고 하는 낡은 판본과 새판본이 있을것이다. 이 새로운 판본을 개정판이라고 한다. 여러개의 판본의 존재는 명백히 낡은 판본은 버리고 새로운것만을 남겨 두도록 함으로써 쉽게 해결되게 된다. 그러나 그것은 아주 어리석은 짓이다. 이전 판본의 모듈이 개정판 n이고 새로운 판본은 개정판 n+1이라고 하자. 개정판 n+1이 쏘프트웨어품질담보(SQA)그룹에 의해서 고립적으로 또는 제품의 나머지부분과 련결시켜 철저히 시험되였더라도 사용자가 새로운 판본의 제품을 실제적인 자료를 가지고 실행할 때는 손해가 막심한 결과가 초래될수도 있다. 개정판 n은 다음의 두번째 리유로 하여 유지정비하여야 한다. 제품은 여러 곳에 분포되였고 그것들은 모두 개정판 n+1을 설치하지 않았을수도 있다. 만일 여전히 개정판 n을 리용하고 있는 곳에서 오유보고서를 보내오면 그때는 이 새로운 오유를 분석하기 위하여 사용자들이 설치한것과 같은 방법으로 정확히 제품을 구성배치하고 그 모듈의 개정판 n을 병합하여야 한다. 그러므로 모든 모듈의 매 개정판의 복제본을 보관해 두는것이 필요하다.

1.3에서 서술한바와 같이 해당한 제품의 기능을 확장하기 위하여 완전유지정비가 진행된다. 일부 경우에는 새로운 모듈들이 작성된다. 즉 기타 다른 경우에 현존 모듈들

은 이런 추가적인 기능을 병합하도록 변경된다. 이런 새로운 판본은 또한 현재의 모듈에 대한 개정판으로 된다. 적응유지정비가 진행될 때 모듈은 변경된다. 즉 제품이 동작하는 주위환경에서의 변화에 대응하여 제품에 대한 변경이 이루어 진다. 문제는 바로 유지정비단계에서 제기되는것이 아니라 실현단계와 그이전단계들에서 발생하기때문에 정확히 유지정비가 진행되면 그이전의 모든 판본들은 유지정비되는것으로 된다. 결국 일단 모듈이 코드작성되면 오유들이 발견되여 정정되는 결과에 계속 변경이 이루어 지게 된다. 결과 매 모듈에 대하여 수많은 판본들이 존재하게 되며 개발팀의 매 성원들이 어느 판본이 어떤 주어 진 모듈에 대한 현재의 판본이라는것을 안다는것을 담보하도록 어떤 종류의 조종을 진행하는것이 사활적인 문제이다. 이 문제에 대한 해결방안이 제기될수 있기전에 그이상의 복잡성이 고려되여야 한다.

5. 7. 2. 변형판

다음의 실례를 고찰하자. 대부분의 콤퓨터는 한가지이상의 인쇄기를 지원하고 있다. 실례로 개인용콤퓨터는 잉크분사식인쇄기와 레이자인쇄기를 지원할수 있다. 그러므로 조작체계는 인쇄기구동프로그람에 대한 두개의 변종을 포함해야 한다. 즉 매 인쇄기에 대하여 하나의 인쇄기구동프로그람을 포함해야 한다. 개정판과는 달리 매개가 그 처리절차프로그람으로 명백히 교체하도록 하는 변형판들이 동시에 존재할수 있도록 설계되여야 한다. 변형판들이 필요하게 되는 또 다른 하나의 상황은 제품이 서로 다른 조작체계나 하드웨어에 대한 변종들에 대하여 처리를 진행하게 될 때이다. 많은 모듈들에 대한 서로 다른 변형판들은 매 조작체계와 하드웨어의 결합을 고려하여 작성되여야 할수도 있다.

변형판들을 구조도식으로 그림 5-10에서 보여 주었는데 여기서는 개정판과 변형판들을 모두 보여 주고 있다.

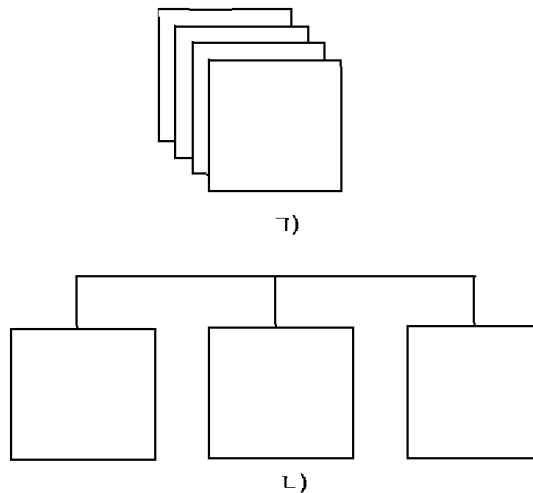



그림 5-10. 여러 판본의 모형에 대한 도식적인 표현
ㄱ) 개정판, ㄴ) 변형판

문제가 더 복잡해 지면 일반적으로 매 변종판에 대하여 여러개의 개정판들이 있게 된다. 쏘프트웨어기업체가 여러개의 판본으로 인한 곤경에 빠져 들어 가는것을 극복하기 위하여 CASE도구가 필요하게 된다.

5. 8. 구성조종

매개 모듈에 대한 코드는 세가지 형식으로 존재하게 된다. 그 하나가 원천코드인데 지금은 보통 C++, Java, Ada와 같은 고급언어로 작성되고 있다. 그다음은 원천코드를 콤파일하여 생성한 목적코드이다. 마지막으로 매개 모듈에 대한 목적코드가 수행적재가능한 프로그람을 생성하기 위하여 실행시간루틴과 결합된다. 이것을 그림 5-11에 보여 주었다.

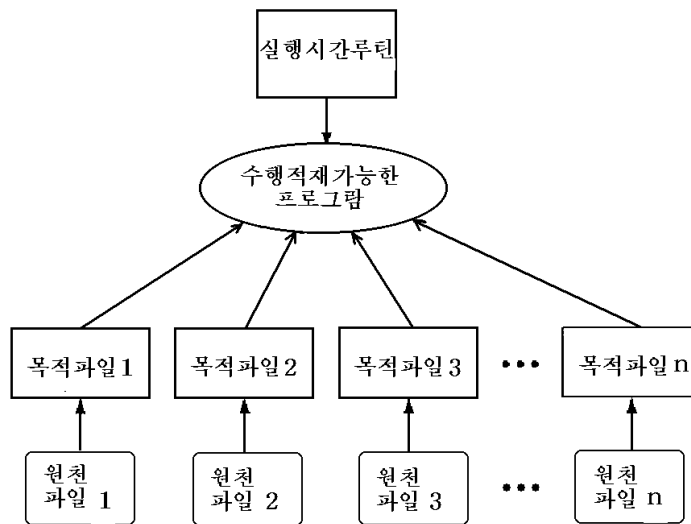



그림 5-11. 수행적재가능한 프로그람의 구성요소

프로그람작성자는 매개 모듈의 여러가지 서로 다른 판본을 리용할수 있다. 완성된 제품의 주어 진 판본이 만들어 지게 되는 그런 매개 모듈의 특정한 판본을 그 제품에 해당한 판본의 구성배치라고 한다. 프로그람작성자에게 어떤 모듈이 특정한 어떤 시험자료모임에 대하여 실패하였다는것을 서술한 SQA그룹의 시험보고서가 제공되였다고 하자. 첫번째로 하여야 할 일들중의 하나는 실패를 다시 생성해 보는것이다. 그러나 어느 변종의 개정판이 파괴된 제품의 판본으로 되겠는가를 프로그람작성자가 어떻게 결정할수 있는가? 만일 구성조종도구(아래에서 론의된다.)를 리용하지 않는다면 오유의 원인을 정확히 지적하기 위한 유일한 방도는 8진수나 16진수형식으로 되여 있는 수행적재가능한 프로그람을 조사하여 그것을 8진수나 16진수로 된 목적코드와 비교해 보는것이다. 특히 여러가지 판본의 원천코드는 콤파일되여 수행적재가능한 프로그람으로 전환된 목적코드와 비교해야 한다. 비록 이렇게 할수 있다고 하더라도 매 제품이 여러개의 판본을 가진 수십(수백은 아니다.)개의 모듈로 이루어 졌다면 이 과정에는 오랜 시간이 걸릴수 있다. 이

로부터 여러개의 판본을 취급할 때 두가지 문제가 해결되여야 한다. 첫째는 매개 모듈에 대한 정확한 판본이 콤파일되여 제품에 련결되도록 판본들사이를 구별해 줄수 있어야 한다는것이다. 둘째 문제는 우와 반대이다. 즉 수행적재가능한 프로그람이 주어 지면 그것을 구성하는 매 요소들의 어느 판본이 그것으로 전환되였는가를 결정해야 한다.

이 문제를 해결하는데 필요한 첫번째 항목은 판본조종도구이다. 많은 조작체계들 특히 대형콤퓨터들에서는 판본조종기능을 지원하고 있다. 그러나 대부분의 체계들에서는 이 기능을 지원하지 않으며 이 경우에는 개별적인 판본조종도구가 필요하다. 판본조종에서 리용된 보편적인 기술은 매 파일의 이름을 두 부분 즉 파일이름 그자체와 개정판수로 구성하는것이다. 이리하여 통보문의 접수를 승인한 모듈은 그림 5-12의 ㄱ)에서처럼 acknowledgeMessage/1, acknowledgeMessage/2 등과 같은 개정판을 가지게 될것이다. 프로그람작성자는 그때 주어 진 과제를 위하여 어느 개정판이 필요한가를 정확히 규정할수 있다.

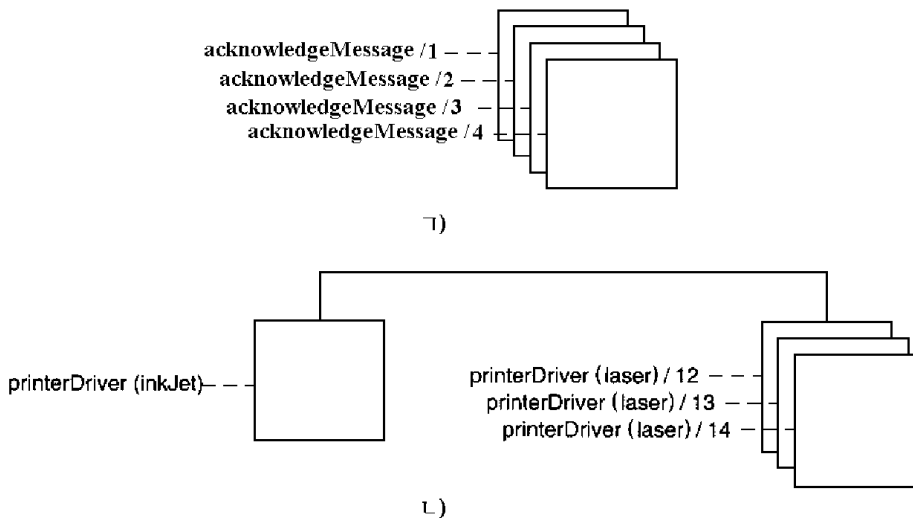



그림 5-12. 여러가지 개정판과 변형판들

ㄱ) 모듈 acknowledgeMessage의 네개의 개정판, ㄴ) 변형판 printerDriver(laser)에 대한 세개의 개정을 가지고 있는 모듈 printerDriver의 두개의 변종

여러개의 변형판(다른 상황에서 같은 역할을 수행하는 판본들을 약간 변화시킨것)들에 대한 하나의 유용한 표기법은 기본적인 파일이름뒤에 괄호안에 넣은 변종이름을 놓는것이다[Babich, 1986]. 이리하여 두개의 인쇄기구동프로그람에 printerDriver(inkJet)와 printerDriver(laser)라는 이름을 주게 된다.

물론 printerDriver(laser)/12, printerDriver(laser)/13, printerDriver(laser)/14와 같이 매 변형판에 대하여 여러개의 개정판이 있게 될것이다. 이것을 그림 5-12의 ㄴ)에서 보여 주었다.

판본조종도구는 여러개의 판본을 관리할수 있도록 하는 첫 걸음으로 된다. 일단 판본조종도구가 설치되면 제품의 여러가지 판본에 대한 상세한 기록(또는 파생물)을 보존하여야 한다. 파생물은 매 원천코드요소의 이름을 포함하고 있다. 이와 함께 변형판과 개

정판 그리고 리용된 여러가지 콤파일러와 련결프로그람의 판본, 제품을 개발한 사람의 이름과 제품이 개발된 날자와 시간 등을 포함하고 있다.

판본조종은 모듈과 제품전반에 대한 여러가지 판본을 관리하는데 큰 도움을 준다. 그러나 여러가지 변형판은 유지와 관련한 추가적인 문제가 제기되는것으로 하여 판본조종이 더욱 필요하게 된다.

두개의 변형판 printerDriver(inkJet)와 printerDriver(laser)에 대하여 고찰해 보자. printerDriver(inkJet)에서 오유가 발견되였고 오유는 두 변형판의 공동적인 모듈의 어느 한 부분에서 발생하였다고 가정하자. 그때는 printerDriver(inkJet)만이 아니라 printerDriver(laser)에 대해서도 보수해야 할 필요가 있다. 일반적으로 하나의 모듈에 대하여 ν개의 변형판이 있다면 ν개의 변형판이 모두 수정되여야 한다. 뿐만아니라 그것들은 다 같은 방법으로 수정되여야 한다. 이 문제에 대한 한가지 해결방안은 printerDriver(inkJet)라고 하는 하나의 변형판을 보관하는것이다. 그다음 임의의 다른 변형판은 원래의 판본으로부터 변형판으로 가서 만들어 져야 할 변경들의 목록에 의하여 보관된다. 이와 같이 차이나는 것들에 대한 목록을 델타(*delta*)라고 부른다. 그러므로 보관된것은 한개의 변형판과 ν-1개의 델타들이다. 변형판 printerDriver(laser)는 printerDriver(inkJet)를 호출하고 델타를 적용하여 회복된다. 바로 printerDriver(laser)에 대하여 만들어 지는 변경은 적당한 델타를 변화시키는것으로써 실현된다. 그러나 원래의 변형판인 printerDriver(inkJet)에 대한 변경은 자동적으로 다른 모든 변형판들에 적용되게 된다.

구성조종도구는 여러개의 변형판들을 자동적으로 관리할수 있다. 그러나 구성조종은 여러개의 변형판들을 초월하여 진행된다. 구성조종도구는 또한 다음 절에서 서술하는바와 같이 팀에 의하여 개발되고 유지정비되는 과정에 제기되는 문제들을 관리할수 있다.

5. 8. 1. 제품유지정비단계에서의 구성조종

한명이상의 프로그람작성자가 동시에 하나의 제품을 유지정비하고 있을 때에는 온갖 종류의 난관이 제기될수 있다. 실례로 월요일 아침에 두명의 프로그람작성자들에게 각각 서로 다른 오유보고서를 준다고 하자. 동시에 두 프로그람작성자는 같은 모듈 mDual에서의 서로 다른 부분에서 수정될 오유들을 나누어 가지게 된다. 매 프로그람작성자는 mDual/16이라는 모듈의 현재판본을 복사하고 오유에 대하여 수정을 시작한다. 첫 사람은 첫 오유를 퇴치하고 변경이 승인되면 그 모듈을 mDual/17이라는 모듈로 교체한다. 그 다음날에 두번째 프로그람작성자는 두번째 오유를 퇴치하고 변경이 승인되면 모듈 mDual/18을 설치한다. 유감스럽게도 개정판 17은 첫 프로그람작성자에 의한 변경만을 포함하고 개정판 18은 다만 두번째 프로그람작성자에 의한 변경만을 포함한다. 이리하여 첫 프로그람작성자에 의한 변경은 모두 잃어 지게 된다.

비록 매 프로그람작성자가 어떤 모듈의 개별적인 복사본을 만들도록 하는 착상이 두 프로그람작성자가 쏘프트웨어의 같은 부분에서 함께 일하는것보다 훨씬 좋다고 할지라도 그것은 명백히 팀에 의하여 유지정비를 진행하는데는 적당치 않다. 필요되는것은 다만 한명의 사용자가 한번에 모듈을 변화시키도록 하는 기구가 필요하다.

5. 8. 2. 기준선

유지정비관리자는 제품에 있는 모든 모듈들에 대한 구성배치(판본의 모임)인 기준선을 설정해야 한다. 오유를 찾으려고 할 때 유지정비프로그람작성자는 필요한 모듈에 대한 복제본들을 개별적인 작업공간에 넣어 준다. 이 개별적인 작업공간에서 프로그람작성자는 임의의 방법으로 다른 프로그람작성자에게 영향을 주지 않는 임의의 모든것을 변화시킬수 있다. 왜냐하면 모든 변경들은 프로그람작성자의 개별적인 복제본에서 만들어 지기때문에 기준판본은 손대지 않은채로 남아 있게 된다.

일단 모듈이 오유를 퇴치하기 위하여 변경되여야 한다는것이 결정되면 프로그람작성자는 교체하려는 모듈의 현재판본을 동결시키게 된다. 다른 모든 프로그람작성자들은 동결시킨 판본을 변경시킬수 없다. 유지정비프로그람작성자가 변경을 진행하고 그 변경들이 시험된 다음 모듈의 새로운 판본이 설치되고 그로 인하여 기준선이 수정된다. 동결된 이전의 판본은 이미 설명한바와 같이 앞으로 필요되기때문에 보존되여야 하지만 교체되지는 않는다. 일단 새로운 판본이 실현되면 그 어떤 다른 유지정비프로그람작성자가 새로운 판본을 동결시키고 거기에 변경을 진행한다. 결과적인 모듈은 차례로 다음의 기준판본으로 되게 된다. 만일 두개이상의 모듈이 동시에 변경되여야 한다면 류사한 처리절차가 진행된다.

이러한 도식은 모듈 mDual에서 제기되는 문제를 해소한다. 두 프로그람작성자는 mDual/16에 대하여 개별적인 복사를 진행하고 그 복제본들을 수정하여야 할 개개의 오유들을 분석하는데 리용한다. 처음의 프로그람작성자는 무엇을 변경시키겠는가를 결정하고 mDual/16을 동결시킨다. 그리고 첫 오유를 퇴치하기 위하여 거기에 변경을 진행한다. 변경이 시험된 다음에 결과적인 개정판 mDual/17을 기준판본으로 설정한다. 한편 두번째 프로그람작성자는 mDual/16에 대한 개별적인 복제본를 가지고 실험함으로써 두번째 오유를 찾아 낸다. 그러나 그것이 첫 프로그람작성자에 의해서 동결되였기때문에 이제는 mDual/16에 대하여 변경이 진행되지 않는다. 일단 mDual/17이 기준판본으로 되면 그것은 mDual/17에 변경을 진행하는 두번째 프로그람작성자에 의해서 동결된다. 결과적인 모듈은 mDual/18 즉 두 프로그람작성자가 협력하여 변경시킨 판본으로서 설치된다. 개정판 mDual/16과 mDual/17은 앞으로 참고하기 위하여 유지정비되지만 절대로 변경될수 없다.

5. 8. 3. 제품개발에서의 구성조종

모듈이 코드작성되는 동안 판본은 너무 빨리 변경되므로 구성조종에 도움을 주지 못한다. 그러나 일단 모듈에 대한 코드작성이 완성되면 그것은 6.6에서 서술한바와 같이 즉시 프로그람작성자에 의하여 비형식적으로 시험되여야 한다. 이러한 비형식적인 시험을 진행하는 기간에 모듈은 또다시 많은 판본들로 변경되게 된다. 프로그람작성자가 동의하면 그것은 규칙적인 시험을 진행하기 위하여 SQA그룹에 넘겨 진다. 모듈은 SQA그룹을 통과하자마자 제품으로 통합될 준비가 된다. 그때로부터 그것은 유지정비단계에서와 같은 구성조종처리절차에 복종되게 된다. 통합된 모듈에 대한 임의의 변경은 유지정비단계에서 변경이 이루어 진 방법으로 제품전반에 대하여 영향을 줄수 있다. 따라서 구성

조종은 유지정비단계에서뿐아니라 실현단계와 통합단계에서도 필요하게 된다. 더우기 매개 모듈이 합리적으로 되자마자 즉 그것이 SQA그룹을 통과한 다음에 구성조종에 복종되지 않으면 관리자측은 개발공정을 충분하게 감시할수 없다. 구성조종이 정확히 적용될 때 관리자는 매 모듈의 상태를 알아 차리고 프로젝트의 최종기한이 다가오는것 같으면 신속하게 정확한 행동을 취할수 있다. UNIX에서 주요한 세개의 판본조종도구로는 sccs(원천코드조종체계; *source code control system*)[Rochkind, 1995]와 rcs(개정판조종체계; *revision control system*)[Tichy, 1985] 그리고 cvs(병행판본체계; *concurrent versions system*)[Loukides and Oram, 1997]이 있다. PVCS는 대중적인 구성조종도구이다. 마이크로쏘프트의 SourceSafe는 개인용콤퓨터를 위한 구성조종도구이다.

5. 9. 구축도구

만일 쏘프트웨어기업체가 완전한 구성조종도구를 구입하려고 하지 않으면 그때는 적어도 구축도구가 판본조종도구 즉 제품의 특정한 판본을 구성하기 위하여 련결되는 매개 목적코드모듈의 정확한 판본을 선택하는것을 도와 주는 도구와 결합하여 리용되여야 한다. 임의의 시간에 매개 모듈에 대한 여러개의 변형판과 개정판들이 제품서고에 있을것이다. 어떤 판본조종도구는 사용자가 원천코드모듈의 여러가지 판본들을 식별해 내는것을 도와 줄것이다. 그러나 일부 판본조종도구는 목적판본에 개정판번호를 붙이지 않기때문에 목적코드를 추적하기가 어렵다.

이에 대처하여 일부 기업체들은 매 모듈의 최신판본을 자동적으로 콤파일하여 모든 목적코드가 항상 갱신되게 한다. 비록 이 기법을 리용해도 불필요한 콤파일이 많이 진행되기때문에 콤퓨터작업시간이 매우 랑비될수 있다. UNIX도구 make는 이 문제를 해결할수 있다[Feldman, 1987]. 매 수행적재가능한 프로그람에 대하여 프로그람작성자는 그림 5-11에서 보여 준 계층과 같은 특정한 구성에 들어 가는 원천 및 목적파일의 계층을 규정하는 Makefile을 설치한다. C나 C++에 있는 포함파일들과 같은 보다 복잡한 의존성들 역시 make에 의해서 조종될수 있다. 프로그람작성자에 의해서 호출될 때 도구는 아래와 같이 동작한다. 사실상 다른 모든 조작체계들과 같이 UNIX는 매개 파일에 날자와 시간을 붙혀 둔다. 원천파일에 대한 표식이 금요일, 7월 6일 오전 11h 24min이고 반면에 대응하는 목적파일에 붙은 표식은 금요일, 7월 6일 오전 11h 40min이라고 가정하자. 그때는 목적파일이 콤파일러에 의해서 창조되였기때문에 원천파일이 변화되여야 한다는것은 명백하다. 다른 한편 만일 원천파일에 있는 날자와 시간표식이 목적파일에 있는 날자와 시간보다 더 늦으면 그때는 make가 원천파일의 현재 판본에 대응한 목적파일의 판본을 창조하기 위하여 적당한 콤파일러나 아쎔블리를 호출한다.

다음에 수행적재가능한 프로그람에 있은 날자와 시간은 그 구성배치에 있는 매개 목적파일에 있는것들과 비교된다. 만일 수행적재가능한 프로그람이 모든 목적파일들보다 더 늦으면 그때는 다시 련결할 필요가 없다. 그러나 목적파일이 수행적재가능한 프로그람보다 더 늦으면 그때는 목적파일에 대한 최신판본을 병합하지 말아야 한다. 이 경우에는 make가 련결프로그람을 호출하여 갱신된 수행적재가능한 프로그람을 구축한다.

다른 말로 make는 수행적재가능한 프로그람이 매개 모듈에 대한 현재의 판본을 병합하는가를 검사한다. 만일 그렇다면 더이상 아무것도 수행되지 않으며 CPU시간이 불필요한 콤파일이나 련결에 랑비되지 않는다. 그러나 그렇지 않다면 그때는 make는 련관된 체계쏘프트웨어를 호출하여 제품의 최신판본을 창조한다.

이밖에 make는 목적파일을 구축하는 과제를 간소화한다. 사용자에게 있어서 어느 모듈이 리용되며 그것들이 어떻게 련결되는가를 매번 명시할 필요는 없다. 왜냐하면 이러한 정보가 이미 Makefile에 있기때문이다. 그러므로 단일한 make지령은 수백개의 모듈을 가진 제품을 구성하며 완성된 제품이 정확하게 조립되였다는것을 담보하기 위하여 필요하다.

5．１０．CASE기술을 리용한 생산성제고

레이훠(문헌 [Myers, 1992]에 제시된바와 같이)는 CASE기술을 도입한 결과로 인한 생산성제고에 대한 연구를 진행하였다. 그는 10여개 산업분야의 45개 회사들로부터 자료를 수집하였다. 회사들가운데서 절반은 정보체계분야와 관련되여 있고 25%는 과학부문, 25%는 실시간우주산업과 관련되여 있다. 해마다 평균생산장성은 9%(실시간우주산업)부터 12%(정보체계)사이에서 변화되였다. 만일 생산성제고만을 고려하게 된다면 이러한 수자는 CASE기술을 도입하여 사용자당 12만 5천딸라의 가격을 얻게 된다는것을 정당화할수는 없다. 그러나 조사된 회사들은 CASE기술에 대한 정당성이 단순히 생산성을 제고할뿐아니라 개발기간을 보다 단축하고 쏘프트웨어의 품질을 개선한다고 보고 있다. 달리 말하면 그것이 비록 일부 CASE기술의 제안자들이 주장한것보다 적다고 하더라도 CASE환경의 도입은 생산성을 제고하였다. 그럼에도 불구하고 쏘프트웨어기업체들이 CASE기술을 도입하는데는 그것이 개발을 더 빨리 하게 하고 오유가 더 적어 지게 하고 유용성을 더 높이며 유지정비를 쉽게 하고 사기를 더 높여주는것과 같은 기타 다른 중요한 원인들이 있다.

15개의 세계 대기업체산하의 500개 회사들에서 진행한 100개이상의 개발프로젝트들로부터 제기된 CASE기술의 효과성에 대한 새로운 결과들은 교육과 쏘프트웨어개발공정의 중요성을 반영하고 있다[Guinan, Cooprider, and Sawyer, 1997]. CASE를 리용한 팀에 특정한 도구에 대한 교육은 물론 일반적인 응용프로그람개발을 위한 교육을 줄 때 사용자들의 요구를 더욱 충족시키게 되였고 개발일정을 맞출수 있었다. 그러나 교육이 진행되지 않으면 쏘프트웨어는 뒤늦게 배포되고 사용자들의 요구는 더욱더 만족시키기 어렵다. 또한 팀이 CASE도구를 구조화방법론과 결합하여 리용하면 성능이 50%까지 올라 갔다. 이러한 결과들은 CASE환경이 성숙준위 1, 2에 있는 그룹에 의하여 리용되지 말아야한다고 한 2.11에서의 주장을 확증해 주고 있다. 사실대로 말한다면 머저리는 도구를 가지고 있어도 여전히 머저리이다[Guinan, Cooprider, and Sawye, 1997].

이 장의 마지막그림인 그림 5-13은 이 장에서 서술한 리론적인 도구들과 CASE도구들을 그것이 서술된 절들과 함께 자모순서로 렬거해 주고 있다.

리론적인 도구
비용 대 리득분석(5.2)
척도(5.3)
계단식세련(5.1)
CASE분류
환경(5.5)
하위CASE도구(5.5)
상위CASE도구(5.5)
작업대(5.5)
CASE도구
구축도구(5.9)
코드작성도구(5.6)
구성조종도구(5.8)
일관성검사기(5.5)
자료사전(5.5)
전자우편(5.6)
대면부검사(5.6)
직결문서작성(5.6)
조작체계앞단처리프로그람(5.6)
인쇄프로그람(5.6)
보고서생성프로그람(5.5)
화면생성프로그람(5.5)
원천준위오유수정프로그람(5.6)
표처리프로그람(5.6)
구조편집기(5.6)
판본조종도구(5.7)
문서처리프로그람(5.6)
WWW열람기(5.6)

그림 5-13. 이 장에서 서술한 리론적인 도구들과 쏘프트웨어 (CASE)도구들 그리고 해당한 절

요 약

첫째로 많은 리론적인 도구들이 있다. 밀러의 법칙에 기초한 계단식세련은 5.1에서 서술하고 례증하였다. 또 하나의 분석도구인 비용 대 리득분석은 5.2에서 서술하였다. 쏘프트웨어척도를 5.3에서 소개하였다.

여러가지 콤퓨터지원쏘프트웨어공학(CASE)도구들은 5.4부터 5.6까지에서 서술하였다. 큰 제품이 개발될 때는 판본조종, 구성조종 그리고 구축도구들이 필수적이다. 이것들은 5.7부터 5.9에서 서술하였다. CASE기술을 리용한 결과로 인한 생산성제고는 5.10에서 서술하였다.

보 충

밀러의 법칙과 관련한 보충적인 정보를 얻기 위하여 그리고 콤퓨터두뇌가 토막들에 대하여 어떻게 동작하는가 하는데 대한 그의 리론을 알기 위하여 독자들은 밀러의 최초의 론문 [Miller, 1956]은 물론 론문 [Tracz, 1979, and Moran, 1981]을 참고할수 있다. 인식원리와 콤퓨터과학의 견지에서의 밀러의 법칙의 분석은 문헌 [Coulter, 1983]에서 찾아 볼수 있다.

계단식세련에 관한 워스의 론문은 이러한 종류의 대표적저서이며 주의 깊은 연구라고 말할수 있다[Wirth, 1971]. 계단식세련의 견지에서 같은 의의를 가지는것은 디지크스트라[Dijkstra, 1976]와 워스[Wirth, 1975]가 쓴 책이다. 밀즈는 계단식세련을 통구조화설계 즉 명세서로부터 설계를 작성하는 기법에 적용하였다[Mills, Linger, and Hevner, 1987; Mills, 1988]. 라즐리츠는 계단식세련을 대규모제품개발에로 확장하였다[Rajlich, 1985]. 실시간체계에 대한 계단식설계는 문헌 [Kurki-Suonio, 1993]에 서술되여 있다.

CASE에 대한 기사는 *IEEE Software* 1995년 3월호와 1996년 9월호에서는 물론 *Communications of the ACM* 1995년 6월호에 소개되였다. 문헌 [Chmura and Crockett, 1995]는 쏘프트웨어개발에서 노는 CASE도구의 역할에 대하여 검토하였다. 도구의 발전에 대한 실례연구는 문헌 [Kitchenham, Pickard, and Pfleeger, 1995]에 서술되여 있다.

이 책에서는 쏘프트웨어개발공정의 개별적인 단계들을 위한 CASE도구들은 매 단계를 설명하고 있는 장들에 서술되여 있다. 작업대나 CASE환경에 대한 정보는 14장의 보충에서 참고할수 있다.

문헌 [Whitgift, 1991]은 구성관리에 대한 좋은 소개도서이다. 쏘프트웨어생명주기의 선택이 구성배치관리에 주는 영향은 문헌 [Bersoff and Davis, 1991]에서 서술하고 있다. 쏘프트웨어구성배치관리에 관한 국제토론회 회보는 구성관리에 대한 정보의 유용한 원천으로 된다.

비용 대 리득분석에 관해서는 문헌 [Gramlich, 1997]을 비롯한 좋은 책들이 많다. 정보체계에 적용된 비용 대 리득분석에 대한 정보는 문헌 [King and Schrems, 1978]을 참고할수 있다. 척도에 관한 중요도서들에는 문헌 [Sheppard 1996, and Fenton and Pfleeger, 1997]들이 포함된다. 아주 상세하게 쓴 책은 [Grady, 1992]이다. 문헌 [Johes, 1994a]는 실현할수 없고 타당치 못함에도 불구하고 문헌에서 계속 언급되여 온 척도들을 강조하고 있다. 객체지향척도는 [Henderson-Sellers, 1996]에 서술되여 있다. 1997

년 3월과 4월에 발표한 *IEEE Software*에는 문헌 [Pfleeger, Jeffrey, Curtis , and Kitchenham, 1997]을 비롯한 척도에 관한 많은 론문들과 쏘프트웨어척도에 대한 평가를 포함하고 있다.

작은 프로젝트에 대한 척도는 *IEEE Software* 1999년 3월/4월호에서 론의하였다. 척도에 관한 많은 기사들은 *IEEE Computer* 1994년 9월호와 *IEEE Software* 1997년 3월/4월호에 있다.

문 제

5.1. 련속적인 주파일갱신문제의 수정된 세번째 세련에 대한 설계에 전망(*lookahead*)을 도입한 효과를 고찰하시오. 즉 하나의 트랜잭션을 실현하기전에 다음트랜잭션이 읽어져야 한다. 만일 두개의 단일트랜잭션이 같은 주파일기록에 적용된다면 현재 트랜잭션의 처리와 관련한 결정은 다음트랜잭션의 류형에 의존하게 된다. 행에는 현재의 트랜잭션의 형을 표시하고 렬에는 다음에 오는 트랜잭션의 형을 표시한 3×3표를 작성하고 매 실례에서 취할 작용을 채워 넣으시오. 실례로 같은 기록에 대한 두개의 련속적인 삽입은 반드시 오유로 된다. 그러나 두개의 변경은 완전히 타당할수 있다. 즉 실례로 신청자는 주어 진 달에 한번이상 주소를 변경시킬수 있다. 이제 전망을 결합한 세번째 세련에 대하여 흐름도를 작성하시오.

5.2. 문제 5.1에 대한 대답이 변경트랜잭션에 뒤이은 삭제트랜잭션을 정확하게 조종할수 있는가를 검토하시오. 이 두 트랜잭션은 동일한 주파일레코드에 적용된다. 만일 그렇지 않다면 대답을 수정하시오.

5.3. 문제 5.1에 대한 대답이 삽입, 변경에 뒤이은 삭제트랜잭션을 정확하게 조종할수 있는가를 검토하시오. 이 트랜잭션들은 모두 동일한 주파일레코드에 적용된다. 만일 그렇지 않다면 대답을 수정하시오.

5.4. 당신의 대답이 n개의 삽입과 수정, 또는 삭제(n>2)를 정확히 조정할수 있는가를 검토하시오. 이 트랜잭션들은 모두 동일한 주파일레코드에 적용된다. 만일 그렇지 않다면 대답을 수정하시오.

5.5. 마지막트랜잭션기록은 뒤에 아무것도 가지고 있지 않다. 그림 5-1에 대한 흐름도가 이것을 고려하고 있으며 마지막트랜잭션기록을 정확히 처리하는가를 검토하시오. 그렇지 않으면 대답을 수정하시오.

5.6. 일부 응용프로그람들에서 전망에 대한 변경은 트랜잭션를 잘 정돈함으로써 실현될수 있다. 실례로 같은 주파일기록에 대한 변경에 뒤이은 삭제에 의하여 야기된다면 원래의 문제는 수정되기전에 삭제를 진행하여 해결될수 있다. 이것은 주파일이 정확하게 작성되고 하나의 오유통보가 례외보고서에 나타나게 할것이다. 문제 5.2, 5.3 그리고 5.4에 렬거된 모든 난점들을 해결할수 있는 트랜잭션의 순서짓기가 존재하는가를 조사

하시오.

5.7. 새로운 형의 위장질병이 콘코르디아(Concordia)나라(역자주; 가상적인 나라)를 휩쓸고 있다. 히스토플라즈마증과 마찬가지로 공기전염된다. 비록 질병이 치명적인것은 아니더라도 그 병균이 침습하면 매우 고통스럽고 약 2주일동안은 일할수 없다. 콘코르디아정부는 아무리 많은 돈이 들더라도 질병을 근절하기 위하여 비용을 소비하기로 결정하려고 한다. 대중건강부서와 의논할 책임을 맡은 위원회는 문제를 네가지 측면 즉 건강관리비용(콘코르디아는 모든 시민들에게 건강관리비용를 무료로 제공하고 있다.), 수입을 잃은것(때문에 세금의 손실), 고통과 불안 그리고 정부에 대한 감사라는 네가지 측면을 고찰하고 있다. 비용 대 리득분석이 위원회를 어떻게 방조할수 있는가를 설명하시오. 얻어 질 가격 또는 리득에 대하여 가격타산을 어떻게 제안하겠는가?

5.8. 1인쏘프트웨어제품개발에서는 판본조종도구를 필요로 하는가? 만일 그렇다면 무엇때문인가?

5.9. 1인쏘프트웨어제품개발기업체에서는 구성조종도구가 필요한가? 만일 그렇다면 무엇때문인가?

5.10. 당신은 소형잠수함의 항해체계를 조종하는 쏘프트웨어를 책임진 관리자이다. 서로 다른 사용자에 의하여 보고된 세개의 오유들을 수정하여야 하는데 그것을 각각 파운, 큐틴, 라첼에게 할당하였다. 하루후에 당신은 세개의 변경을 모두 실현하기 위해서는 동일한 4개의 모듈이 변경되여야 한다는것을 알았다. 그러나 당신의 구성조종도구는 조작을 하지 않는다. 그러므로 당신은 자체로 변경들을 관리해야 할것이다. 그것을 어떻게 할것인가?

5.11. 2.11에서는 성숙준위 1, 2에 있는 기업체내에서 CASE환경을 도입하는것은 거의 의미가 없다는것을 언급하였다. 이것이 왜 그런가를 설명하시오.

5.12. 낮은 성숙준위에 있는 기업체내에 CASE도구(개발환경에 반대되는)를 도입한 효과는 무엇인가?

5.13. (과정안상 목표) 어떤 류형의 CASE도구가 부록 1에서 서술한 브로드랜즈지역아동병원제품을 개발하는데 적합한가?

5.14. (쏘프트웨어공학독본) 교원이 문헌 [Wirth, 1971]에 대한 복제본을 배포할것이다. 워스의 방법과 이 장에서 서술한 계단식세련방법사이의 차이를 렬거하시오.

참 고 문 헌

[Apple, 1984] *Macintosh Pascal User's Guide*, Apple Computer, Inc., Cupertino, CA, 1984.

[Babich, 1986] W. A. Babich, *Software Configuration Management: Coordination for Team Productivity*, Addison-Wesley, Reading, MA, 1986.

[Bersoff and Davis, 1991] E. H. Bersoff and A. M. Davis, "Impacts of Life Cycle Models on Software Configuration Management," *Communications of the ACM* **34** (August 1991), pp. 104–18.

[Chmura and Crockett, 1995] A. Chmura and H. D. Crockett, "What's the Proper Role for CASE Tools?" *IEEE Software* **12** (March 1995), pp. 18–20.

[Coulter, 1983] N. S. Coulter, "Software Science and Cognitive Psychology," *IEEE Transactions on Software Engineering* **SE-9** (March 1983), pp. 166–71.

[DeRemer and Kron, 1976] F. DeRemer and H. H. Kron, "Programming-in-the-Large versus Programming-in-the-Small," *IEEE Transactions on Software Engineering* **SE-2** (June 1976), pp. 80–86.

[Dijkstra, 1976] E. W. Dijkstra, *A Discipline of Programming*, Prentice Hall, Englewood Cliffs, NJ, 1976.

[Dooley and Schach, 1985] J. W. M. Dooley and S. R. Schach, "FLOW: A Software Development Environment Using Diagrams," *Journal of Systems and Software* **5** (August 1985), pp. 203–19.

[Feldman, 1979] S. I. Feldman, "Make—A Program for Maintaining Computer Programs," *Software—Practice and Experience* **9** (April 1979), pp. 225–65.

[Fenton and Pfleeger, 1997] N. E. Fenton and S. L. Pfleeger, *Software Metrics: A Rigorous and Practical Approach*, 2nd ed., IEEE Computer Society, Los Alamitos, CA, 1997.

[Fuggetta, 1993] A. Fuggetta, "A Classification of CASE Technology," *IEEE Computer* **26** (December 1993), pp. 25–38.

[Grady, 1992] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[Gramlich, 1997] E. M. Gramlich, *A Guide to Benefit–Cost Analysis*, 2nd ed., Waveland Books, Prospect Heights, IL, 1997.

[Guinan, Cooprider, and Sawyer, 1997] P. J. Guinan, J. G. Cooprider, and S. Sawyer, "The Effective Use of Automated Application Development Tools," *IBM Systems Journal* **36** (No. 1, 1997), pp. 124–39.

[Henderson-Sellers, 1996] B. Henderson-Sellers, *Object-Oriented Metrics: Measures of Complexity*, Prentice Hall, Upper Saddle River, NJ, 1996.

[Jones, 1994a] C. Jones, "Software Metrics: Good, Bad, and Missing," *IEEE Computer* **27** (September 1994), pp. 98–100.

[King and Schrems, 1978] J. L. King and E. L. Schrems, "Cost–Benefit Analysis in Information Systems Development and Operation," *ACM Computer Surveys* **10** (March 1978), pp. 19–34.

[Kitchenham, Pickard, and Pfleeger, 1995] B. Kitchenham, L. Pickard, and S. L. Pfleeger, "Case Studies for Method and Tool Evaluation," *IEEE Software* **12** (July 1995), pp. 52–62.

[Kurki-Suonio, 1993] R. Kurki-Suonio, "Stepwise Design of Real-Time Systems," *IEEE Transactions on Software Engineering* **19** (January 1993), pp. 56–69.

[Loukides and Oram, 1997] M. K. Loukides and A. Oram, *Programming with GNU Software*, O'Reilly and Associates, Sebastopol, CA, 1997.

[Miller, 1956] G. A. Miller, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review* **63** (March 1956), pp. 81–97.

[Mills, 1988] H. D. MILLS, "Stepwise Refinement and Verification in Box-Structured Systems," *IEEE Computer* **21** (June 1988), pp. 23–36.

[Mills, Linger, and Hevner, 1987] H. D. MILLS, R. C. LINGER, AND A. R. HEVNER, "Box Structured Information Systems," *IBM Systems Journal* **26** (No. 4, 1987), pp. 395–413.

[Moran, 1981] T. P. MORAN (Editor), Special Issue: The Psychology of Human-Computer Interaction, *ACM Computing Surveys* **13** (March 1981).

[Myers, 1992] W. MYERS, "Good Software Practices Pay Off—or Do They?" *IEEE Software* **9** (March 1992), pp. 96–97.

[Pfleeger, Jeffrey, Curtis, and Kitchenham, 1997] S. L. PFLEEGER, R. JEFFREY, B. CURTIS, AND B. KITCHENHAM, "Status Report on Software Measurement," *IEEE Software* **14** (March/April 1997), pp. 33–44.

[Rajlich, 1985] V. RAJLICH, "Stepwise Refinement Revisited," *Journal of Systems and Software* **5** (February 1985), pp. 81–88.

[Rochkind, 1975] M. J. ROCHKIND, "The Source Code Control System," *IEEE Transactions on Software Engineering* **SE-1** (October 1975), pp. 255–65.

[Shepperd, 1996] M. SHEPPERD, *Foundations of Software Measurement,* Prentice Hall, Upper Saddle River, NJ, 1996.

[Sobell, 1995] M. G. SOBELL, *A Practical Guide to the UNIX System,* 3rd ed., Benjamin/Cummings, Menlo Park, CA, 1995.

[Tichy, 1985] W. F. TICHY, "RCS—A System for Version Control," *Software—Practice and Experience* **15** (July 1985), pp. 637–54.

[Tracz, 1979] W. J. TRACZ, "Computer Programming and the Human Thought Process," *Software—Practice and Experience* **9** (February 1979), pp. 127–37.

[Whitgift, 1991] D. WHITGIFT, *Methods and Tools for Software Configuration Management,* John Wiley and Sons, New York, 1991.

[Wirth, 1971] N. WIRTH, "Program Development by Stepwise Refinement," *Communications of the ACM* **14** (April 1971), pp. 221–27.

[Wirth, 1975] N. WIRTH, *Algorithms + Data Structures = Programs,* Prentice Hall, Englewood Cliffs, NJ, 1975.

[Yourdon, 1989] E. YOURDON, *Modern Structured Analysis,* Yourdon Press, Englewood Cliffs, NJ, 1989.

제6장. 시　험

쏘프트웨어생명주기모형들은 모두 통합단계후와 유지정비단계전에 흔히 개별적인 시험단계를 포함하게 된다. 품질이 좋은 쏘프트웨어를 개발하려고 하는 견지에서 이것은 전혀 위험한 일은 아니다. 시험은 쏘프트웨어개발공정의 중요한 구성요소이며 생명주기 전반에 걸쳐서 수행되여야 할 활동이다. 요구사항확정단계에서는 요구사항이 검열되여야 하고 명세작성단계에서는 명세가 검열되여야 하며 쏘프트웨어생산관리계획은 이와 류사한 정밀조사를 거쳐야 한다. 설계단계는 모든 단계들에서 주의 깊게 검열할것을 요구한다. 코드작성단계에서는 매 모듈들이 반드시 시험되여야 하며 제품전체는 통합단계에서 시험되여야 한다. 인수시험을 거친후에 제품은 설치되고 유지정비단계가 시작된다. 유지정비와 밀접히 결부되여 변경된 제품판본들에 대한 반복적인 검열이 진행된다.

달리 말하면 단순히 그 단계의 마감에서 그 단계의 결과물을 시험하는것은 충분하지 못하다. 실례로 명세작성단계를 고찰하자. 명세작성팀성원들은 자기들이 개발하는 동안 명세서를 자각적으로 량심적으로 검열해야 한다. 개발팀이 몇주 또는 몇달후에 공정의 초기단계에서 만들어 진 오유가 거의 모든 명세서를 다시 작성할것을 요구한다는것을 찾아 내기 위한 목적으로만 완전한 명세서를 개발하는것은 그리 쓸모가 없다. 그러므로 매 단계의 마감에 보다 정연한 시험을 진행하는것을 비롯하여 매 단계를 진행하는 동안 개발팀이 끊임없이 시험을 진행하는것이 필요하다.

> **알고 싶은 문제**
>
> 배리 보엠(Barry Boehm)은 검증과 타당성에 대하여 다음과 같이 정의하였다[Boehm, 1984a].
> 검　증: 제품이 옳바로 구성되고 있는가?
> 타당성: 정확한 제품이 구성되고 있는가?

용어 검증(*verification*)과 타당성(*validation*)은 2장에서 소개되였다. 검증은 해당한 단계가 정확히 수행되였는가를 결정하기 위한 공정으로 간주된다. 즉 이것은 매 단계의 마감에 진행된다. 한편 타당성은 제품이 의뢰자에게 배포되기전에 진행되는 강력한 평가공정이다. 그 목적은 제품전체가 명세서를 만족시키는가를 결정하는것이다(약간 다른 정의를 알고 싶으면 우의 《알고 싶은 문제》를 보시오.). 이 두 용어는 *IEEE Software Engineering* 용어소사전 [IEEE 610.12, 1990]에서 이런 방법으로 정의되였지만 시험을 의미하는 용어 V&V가 일반적으로 시험을 지적하기 위하여 리용됨에도 불구하고 단어 《검증》과 《타당성》은 이 책에서 될수록 적게 리용된다. 그 하나의 리유는 6.5에서 설명한바와 같이 단어 《검증》이 시험의 범위내에서 다른 의미를 가지기때문이다. 두번째 리유는 《검증과 타당성》(또는 V&V; verification and validation)이라는 말이 어떤 단계를 검열하는 공정이 그 단계의 끝까지 연기될수 있다는것을 의미하기때문이다. 반

대로 이러한 검열이 모든 쏘프트웨어개발과 유지정비활동과 병렬로 수행된다는것이 본질적으로 중요하다. 그러므로 V&V이라는 말의 요구되지 않는 내포적의미를 피하기 위하여 용어 시험(*testing*)이 리용된다.

본질에 있어서 두가지 류형의 시험 즉 실행에 기초한 시험과 비실행에 기초한 시험이 존재한다. 실례로 작성된 명세서를 실행하는것은 불가능하다. 즉 유일한 대안은 명세서를 될수록 주의깊게 심사하든가 그것에 대하여 어떤 형식의 분석을 진행하는것이다. 그러나 일단 실행가능한 코드가 있으면 그것은 시험실례를 실행시킬수 있으며 즉 그것은 실행에 기초한 시험으로 된다. 그럼에도 불구하고 코드의 존재는 비실행에 기초한 시험을 배제하지 않는다. 왜냐하면 앞으로 설명되는바와 같이 코드를 주의 깊게 심사하는것은 적어도 시험실레를 동작시킬 때만 한 량의 오유들을 로출시키게 되기때문이다. 이 장에서는 실행에 기초한 시험과 비실행에 기초한 시험에 대한 원리를 서술하고 있다. 이 원리들은 10장부터 16장까지에서 적용되는데 여기서는 공정모형의 매 단계와 그것에 응용할수 있는 특정한 시험실천들에 대하여 서술한다. 이 책의 첫 《알고 싶은 문제》에서 서술한 오유들은 치명적인 결과를 초래한다. 다행히도 대부분의 경우에 잔류오유를 가진 쏘프트웨어를 배포하는 경우에 피해가 그리 크지 않다. 그러나 시험의 중요성을 지나치게 강하게 강조할수는 없다.

6．1．품질문제

용어 품질(*quality*)은 흔히 쏘프트웨어범위안에서 리용될 때 잘못 리해되게 된다. 결국 품질은 어떤 종류의 우수성을 의미하지만 이것은 유감스럽게도 쏘프트웨어공학자들이 의도하는 의미는 아니다. 사실대로 말하면 쏘프트웨어개발에서의 세련된 기교는 순전히 정확하게 기능을 수행하는 쏘프트웨어를 얻으면 충분하다는것이다. 즉 우수성은 우리가 보통 현재의 쏘프트웨어기법으로 가능한것보다 높은 준위에 있다는것을 의미한다. 쏘프트웨어의 품질은 제품이 명세서를 만족시키는 정도를 의미하고 있다(다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

용어 품질을(《우수한》또는《훌륭한》과 대립하여) 《명세서에 충실하라.》라는 의미로 리용하는것은 공학 및 제조업과 같은 분야에서 현실적이다. 실례로 코카콜라병생산기업소에서 품질조종관리자들을 고찰해 보자. 품질조종관리자의 업무는 생산흐름선에 따라 생산되는 병들이나 깡통들이 모든 면에서 코카콜라에 대하여 명세서를 만족시킨다는것을 담보하는것이다. 《우수한》 코카콜라 혹은 《훌륭한》 코카콜라를 생산하려는 시도는 그 어디에도 없다. 즉 유일한 목적은 바로 매개 코카콜라병이나 깡통이 그 탄산음료에 대한 규정한 회사의 규정(명세)를 엄밀히 준수하고 있다는것을 확증하는것이다.

용어 품질은 자동차공업에서도 마찬가지로 리용된다. 품질이 첫째라는것은 지난날 포드자동차회사가 내세웠던 구호이다. 다른 말로 포드의 목적은 포드생산흐름선에서 나온 매 승용차가 그 차에 대한 명세서를 철저히 준수하고 있다는것을 담보하는것이다. 즉 일반적인 쏘프트웨어공학의 의미에서 말한다면 승용차는 모든 면에서 《바그》가 없어야 한다는것이다.

매 쏘프트웨어전문가들의 과제는 항상 품질이 좋은 쏘프트웨어를 담보하는것이다. 그러나 쏘프트웨어품질보증(SQA)그룹은 쏘프트웨어품질과 관련한 추가적인 책임을 지고 있다.

6. 1. 1. 쏘프트웨어품질보증

SQA그룹의 한가지 임무는 제품이 정확하다는것을 담보하는것이다. 더 정확하게는 일단 개발자가 어떤 단계를 완성하면 SQA그룹성원들은 그 단계가 정확히 진행되였다는 것을 검열해야 한다. 또한 제품이 완성될 때 SQA그룹은 그 제품이 전체적으로 정확하다는것을 검열해야 한다. 그러나 쏘프트웨어품질보증는 해당한 단계의 마감이나 개발공정의 마감에 진행하는 시험(V&V)보다 더 앞서 진행된다. 즉 SQA는 쏘프트웨어개발공정 그자체에 적용된다. 실례로 SQA그룹의 책임에는 쏘프트웨어가 준수해야 할 여러가지 규격들의 개발은 물론 이 표준이 준수되고 있다는것을 담보하는 감시절차를 확립하는것이 포함된다. 간단히 말하여 SQA그룹의 역할은 품질이 좋은 쏘프트웨어개발공정을 담보하고 그로 하여 품질이 좋은 제품을 담보하는것이다.

6. 1. 2. 관리독립성

개발팀과 SQA그룹사이의 관리독립성(*managerial independence*)이 중요한 문제이다. 즉 개발은 한명의 관리자에 의해서 관리된다. 그리고 SQA는 또 다른 관리자의 관리하에 있으며 어느 관리자도 다른 관리자를 지배할수 없어야 한다. 그 리유는 심각한 오유들이 모두 배포최종기한이 다가옴에 따라 너무 자주 제품에서 발견되기때문이다. 쏘프트웨어기업체는 이제 만족스럽지 못한 두가지 선택가운데서 하나를 선택해야 한다. 즉 제품이 제시간에 실현될수 있지만 오유가 매우 많아서 의뢰자들이 오유쏘프트웨어와 씨름질을 하게 하든가 또는 개발자들이 그 쏘프트웨어를 수정할수 있지만 그것을 늦게 배포하든가 하는 두가지 경우가 있다. 어쨌든 의뢰자들은 그 쏘프트웨어기업체를 믿지 않게 될것이다. 개발을 책임진 관리자는 오유쏘프트웨어를 제시간에 배포할데 대한 결정을 채택하지 말아야 하며 또한 SQA관리자는 이후의 시험을 수행하고 제품을 늦게 배포할데 대한 결정을 채택할수 없어야 한다. 대신에 두 관리자는 두개의 선택가운데서 어느것이 쏘프트웨어기업체와 의뢰자들에게 가장 흥미 있는가를 결정할 권한을 가지고 있는 보다 상급인 관리자에게 보고해야 한다. 얼핏 보면 개별적인 SQA그룹을 가지는것은 쏘프트웨어개발에 드는 비용이 상당히 추가되는것으로 된다. 그러나 사실은 그렇지 않다. 추가적인 비용은 결과적인 리득 즉 품질이 좋은 쏘프트웨어에 비해서 상대적으로 작다. SQA그룹이 없으면 쏘프트웨어개발기업체의 매 성원들은 품질보증과 관련한 활동에 일정한 정도로 망라되여야 한다. 어떤 개발기업체가 100명의 쏘프트웨어전문가들을 가지고 있는데 매 사람들은 자기 작업시간의 30%를 품질보증활동에 바친다고 가정하자. 이대신에 100명의 성원들을 두개의 그룹 즉 쏘프트웨어개발을 진행하는 70명의 성원들로 이루어 진 그룹과 SQA를 책임진 30명의 성원들로 이루어 진 그룹으로 가른다. SQA에 같은 량의 시간이 돌려 지며 SQA그룹을 지도하는 관리자에게만 추가적인 비용이 소비된다. 이제는 품질보

증과 독립적인 전문가들의 그룹에 의해 수행될수 있는데 이것은 SQA활동의 기업체전반에서 수행될 때보다 품질이 더 높은 제품을 개발할수 있게 한다.

매우 작은 쏘프트웨어회사(4명의 직원 또는 그보다 더 적은 직원을 가진)의 경우에는 개별적인 SQA그룹을 가지는것이 단순히 경제적견지에서 리익이 없을수도 있다. 이러한 환경에서 할수 있는 가장 좋은 방도는 명세서가 이 명세서를 작성하는데 책임이 있는 사람이 아닌 다른 사람에 의해서 검열되고 설계작성, 코드작성 등에 대해서도 이와 류사한 방식으로 검열된다는것을 담보하는것이다. 그 리유를 다음절에서 설명한다.

6. 2. 비실행에 기초한 시험

문서를 작성해야 할 책임을 지닌 사람을 그에 대한 심사를 책임진 유일한 사람으로 간주하는것은 좋은 생각이 아니다. 거의 모든 사람들은 어떻게 오유가 문서에 끼여 들어가는가를 잘 리해할수 없으며 따라서 오유가 심사에서 왜 발견되지 않는가도 잘 리해할수 없다. 그러므로 심사과제는 문서의 원래작성자가 아니라 다른 사람이 진행하여야 한다. 이밖에 오직 한명이 심사를 하는것은 충분하지 못하다. 즉 우리는 문서를 여러번 읽고서도 다른 사람들이 즉시에 찾아 내는 뻔한 맞춤법오유를 발견해 내지 못하는 경우들을 체험하였다. 이것이 바로 관통심사회의나 검토와 같은 심사기법의 기초를 이루고 있는 원리이다. 이 두가지 류형의 심사에서 문서(명세서나 설계문서와 같은)는 폭 넓은 기능을 가지고 있는 쏘프트웨어전문가들의 팀에 의해서 주의 깊게 검열된다. 전문가팀에 의한 심사의 우월성은 각이한 기능을 가진 동업자들이 오유를 발견할 기회를 증대시킨다는것이다. 이밖에 공동작업하는 기능 높은 사람들로 이루어 진 팀은 흔히 긍정적인 효과를 나타낸다.

관통심사회의나 검토는 심사의 두 류형으로 된다. 그것들사이의 근본적인 차이는 관통심사회의가 검토보다 더 낮은 단계이고 덜 형식적이라는것이다.

6. 2. 1. 관통심사회의

관통심사회의를 진행하는 팀은 4～6명으로 구성할수 있다. 명세서관통심사회의팀은 적어도 한명의 명세작성팀대표, 명세작성에 책임이 있는 관리자, 의뢰자대표, 개발의 다음단계(이 경우에는 설계단계)를 수행하게 될 팀의 대표 그리고 쏘프트웨어품질보증그룹의 대표로 구성된다. 다음절에서 설명하는것처럼 SQA그룹성원들은 관통심사회의에 참가하게 된다. 관통심사회의팀성원들은 중요한 결함들을 찾아야 하기때문에 높은 기술과 경험을 가진 성원들로 되여야 한다. 즉 그들은 프로젝트들에 부정적인 영향을 주는 결함들을 찾게 된다[New, 1992].

관통심사회의를 위한 자료는 관계자들이 준비할수 있도록 미리 배포되여야 한다. 매 심사자들은 자료를 연구하고 다음의 두 목록 즉 검토하는 사람들이 리해하지 못하는 항목들의 목록과 심사자들이 부정확하다고 생각하는 항목들의 목록을 작성한다.

6. 2. 2. 관통심사회의의 운영

만일 관통심사회의가 잘 진행되지 않고 결함이 계속 나오면 SQA대표는 큰 손실을 입기때문에 관통심사회의는 SQA대표가 사회한다. 이와 대조적으로 명세작성단계를 책임진 대표는 기타 다른 과제들에 착수하기 위하여 될수록 빨리 명세서들이 승인되기를 바랄수 있다. 의뢰자대표는 검토에서 발견되지 않는 일부 오유들이 인수시험단계에서 나타나게 되며 따라서 의뢰자측에 더는 비용을 들이지 않고 수정된다고 결정할수 있다. 그러나 SQA대표는 가장 큰 리해관계를 가지고 있다. 즉 제품의 품질은 SQA그룹의 전문가적능력에 대한 직접적인 반영이다. 관통심사회의를 사회하는 사람은 오유들을 발견하기 위하여 관통심사회의팀의 다른 성원들이 그 문서를 보도록 안내해 준다. 오유를 정정하는것은 팀의 과제가 아니며 순전히 그후의 정정을 위하여 그것들을 기록해 두는것이다. 이에 대해서는 네가지 리유가 있다.

1. 위원회(즉 관통심사회의팀)가 제한된 관통심사회의시간내에 진행한 정정은 필요한 기법들을 습득한 개별적인 성원들이 진행한 정정보다 품질이 좋지 못한것 같다.
2. 5명의 성원들로 이루어 진 관통심사회의팀이 진행한 정정은 적어도 개별적인 사람들이 진행한 정정만한 시간이 걸린다. 그러므로 5명의 참가자들에게 지불되는 로임을 생각할 때 비용은 5배나 많이 든다.
3. 오유이라고 표식한 모든 항목들이 사실상 부정확한것은 아니다. 《만일 그것이 쪼개지지 않으면 그것은 수정할수 없다.》라는 주장에 따라서 팀이 완전히 정정된 것을 《수정》하려고 하는것이 아니라 오유에 대하여 잘 분석하고 실제적으로 문제가 있을 때에만 정정하는것이 더 낫다.
4. 관통심사회의에서는 오유를 발견하고 정정하기 위한 충분한 시간이 없다. 즉 관통심사회의는 2h이상 하지는 않는다. 오유를 찾고 기록하는데만 시간을 들이며 오유를 정정할 시간은 없다.

관통심사회의를 지도하는데는 두가지 방법이 있다.

첫번째는 어떤 관계자가 지도하는것이다. 관계자들은 명백치 않은 항목들과 그들이 부정확하다고 생각하는 항목들의 목록을 제시한다. 명세작성팀대표는 심사자들에게 무엇이 명백하지 않은가를 밝히며 실지 오유가 존재한다는데 동의하든가 또는 심사자들이 무엇때문에 실수했는가를 설명해 주면서 매개의 질문들에 대답하여야 한다.

검토를 지도하는 두번째 방법은 문서를 가지고 진행하는것이다. 그가 개별적이든 팀의 성원이든 문서작성을 책임진 사람은 검토자들이 자기들이 준비한 설명문이든가 선전물에 의한 설명문을 가지고 참견을 하면서 관계자들이 그 문서를 훑어 보게 한다. 이 두번째 방법이 좀더 철저할수 있다. 이밖에 문서를 가지고 진행하는 관통심사회의에서 제기되는 기본 오유들은 제출자에 의해서도 발견되기때문에 이 방법은 보다 많은 오유들을 발견하게 한다. 제출자들이 매 문장을 자주 읽는 과정에 여러번 읽을 때도 드러나지 않았던 오유들이 별안간 명백하게 될수도 있다. 심리학자들이 진행하는 어떤 유익한 연구는 명세관통심사회의와 설계관통심사회의, 계획관통심사회의, 코드관통심사회의들을 비롯한 모든 종류의 관통심사회의를 진행하는 기간에 언어적서술에 의해서 흔히 오유가 발견되는 리유를 밝혀 줄것이다.

놀랍지 않게 보다 철저한 문서에 의한 심사는 쏘프트웨어검토 및 심사에 대한 IEEE규격(IEEE Standard for Software Reviews and Audits)[IEEE 1028, 1997]에서 규정된 기법이다.

관통심사회의 사회자의 중요한 역할은 질문을 유도하고 토론분위기를 조성하는것이다. 관통심사회의는 대화과정이다. 즉 제출자 한쪽측만이 지시하도록 하는것은 금물이다. 관통심사회의가 관계자들을 평가하는 수단으로 리용되지 않는다는것도 중요한 문제이다. 만일 그런 일이 발생한다면 회의사회자가 아무리 잘하려고 노력한다고 하더라도 관통심사회의는 점수따기식회의로 변질되여 오유들을 발견하지 못하게 된다. 검토되고 있는 문서를 책임진 관리자는 추천된 관통심사회의팀의 한 성원이다. 만일 이 관리자 역시 관통심사회의의 성원들(특히는 제출자)에 대한 년중사업평가에 책임이 있다면 팀의 오유발견능력은 약화되게 될것이다. 왜냐하면 제출자의 원래의 동기는 발견되는 오유의 수를 최소화하는것이기때문이다. 이러한 흥미 있는 모순을 방지하기 위하여 어떤 주어 진 단계를 책임진 사람은 그 단계를 위한 임의의 관통심사회의팀성원을 평가할 책임을 직접 지지 말아야 한다.

6. 2. 3. 검 토

검토는 설계와 코드를 시험할 목적으로 파간이 처음으로 제안하였다[Fagan, 1976].

검토는 관통심사회의를 훨씬 초월하며 5개의 형식적인 단계를 가지고 있다. 첫째로 검토될 문서(명세서, 설계, 코드 또는 계획)에 대한 개괄(*overview*)은 그 문서작성을 책임진 어느 한 사람에 의해서 작성된다. 개괄회의마감에 문서는 관계자들에게 배포된다. 둘째 단계 즉 준비단계에서 관계자들은 문서들을 상세하게 리해하려고 애 쓴다. 빈도별로 정렬된 최근의 검토에서 발견된 오유류형들에 대한 목록은 훌륭한 수단으로 된다. 이러한 목록들은 팀성원이 오유가 제일 많이 발생할수 있는 부분들에 집중할수 있게 한다. 셋째 단계는 검토(*inspection*)이다. 시작되면 한명의 관계자는 검토팀성원과 함께 문서들을 쭉 훑어 보고 매 항목들이 포함되여 있으며 매 부분들이 적어도 한번 취해 진다는것을 담보한다. 그다음 오유찾기가 개시된다. 관통심사회의방법에서와 마찬가지로 목적은 오유를 찾고 문서작성하는것이지 그것들을 정정하는것은 아니다. 그날중으로 검토팀책임자(중재자)는 수행이 세심히 수행되고 있다는것을 담보하기 위한 검토에 대한 서면보고서를 작성한다. 넷째 단계는 재작업(*rework*)인데 여기에서는 그 문서를 책임진 사람이 서면보고서에서 지적된 모든 오유들과 문제들을 해결한다. 마지막단계는 뒤조사(*follow-up*)이다. 중재자는 문서들을 퇴치하기도 하고 오유로 잘못 표시된 항목들을 명백히 함으로써 제기된 매개의 단순한 문제점들이 만족하게 해결되였다는것을 담보해야 한다. 그 어떤 새로운 오유가 인입되지 않았다는것을 담보하기 위하여 모든 수정결과들이 검열되여야 한다[Fagan, 1986]. 만일 검토된 자료의 5%이상이 재작업되였다면 팀은 100% 재검토하기 위하여 다시 소집되여야 한다.

검토는 4명으로 이루어 진 팀에 의해서 지도되여야 한다. 실례로 설계검토의 경우에 팀은 중재자, 설계자, 실현자, 시험자로 구성된다. 중재는 검토팀에서 관리자로도 되고 책임자로도 된다. 다음단계를 책임진 팀의 대표는 물론 현단계를 책임진 대표도 있어야 한다. 설계자는 설계를 작성하는 팀의 한 성원이며 반면에 실현자는 개별적으로든지 팀의 한 부분으로든지 설계를 코드로 바꾸는것을 책임진다. 파간은 시험자가 시험실례를 보장

하는것을 책임진 어떤 프로그람작성자이라고 제기하였다. 물론 시험자는 SQA그룹의 한 성원으로 되는것이 더 낫다. IEEE규격은 3～6명의 관계자들로 이루어 진 팀을 구성할것을 권고하였다[IEEE 1028, 1997]. 중재자는 특정한 역할을 논다. 즉 그는 팀을 설계에로 이끌어 나가는 책임자(*leader*)이며 또한 발견된 오유에 대한 서면보고서를 작성하는것을 책임진 기록자(*recorder*)이기도 하다.

검토에서 필수적인 구성요소는 가능한 오유들에 대한 검열목록이다. 실례로 설계검토에 대한 검열목록은 다음과 같은 항목들을 포함한다. 즉 명세서의 매개 항목이 충분하고도 정확하게 제기되였는가? 매개 대면부들에 대하여 실제의 인수와 형식적인 인수가 대응되는가? 오유조종기구가 적당하게 결합되고 있는가? 설계가 하드웨어자원과 호환가능한가 또는 설계가 실제적으로 리용되는것이상으로 하드웨어를 요구하지 않는가? 설계가 쏘프트웨어자원과 호환가능한가? 례하면 명세서에 규정된 조작체계가 설계에서 요구하고 있는 기능을 가지고 있는가? 하는것이다. 검토절차에서 중요한 구성요소는 오유통계에 대한 기록이다. 오유는 엄격하게(중요성 또는 부차성에 따라서 중요한 오유의 한가지 실례는 자료기지의 조기종결 또는 손상을 초래하는 오유이다.) 오유류형별로 기록하여야 한다. 설계검토의 경우에 전형적인 오유류형에는 대면부오유와 론리적오유가 포함된다. 이러한 정보는 많은 유용한 방식으로 리용될수 있다.

1. 주어 진 제품에서의 오유개수는 대비되는 제품에서 같은 개발단계에서 발견된 오유의 평균값과 비교될수 있으며 관리자에게 무엇인가 잘못되였다고 경고를 주고 시기적절하게 오유정정작업이 진행되게 한다.
2. 2개 또는 3개의 모듈에 대한 설계검토를 진행하여 특정한 류형의 오유가 불균형적인 개수를 가진다는것을 발견하게 되면 관리자는 다른 모듈들에 대한 검열을 시작할수 있으며 오유정정작업을 진행한다.
3. 만일 특정한 모듈에 대한 설계검토에서 제품에 있는 임의의 다른 모듈에서 발견된것보다 훨씬 더 많은 오유들이 발견되면 보통 그 모듈을 처음부터 다시 설계해야 한다.
4. 설계검토에서 발견된 오유의 개수와 류형에 대한 정보는 팀으로 하여금 뒤단계에서 같은 모듈에 대한 코드검토를 진행할수 있게 한다.

파간의 첫번째 실험은 체계제품에 대하여 진행되였다[Fagan, 1976]. 4명으로 된 팀이 하루에 두 사람이 2h 검토하는 속도로 백명의 사람들이 1h동안 검토를 진행하게 하였다. 제품이 개발되는 기간에 발견된 모든 오유들가운데서 67%는 모듈시험이 시작되기전에 검토에 의하여 발견되였다. 더우기 제품이 설치된 다음에 첫 7개월동안 비형식적인 관통심사회의를 리용하여 심사된 비교되는 제품에서보다 검토된 제품에서는 38% 더 적은 오유가 발견되였다.

파간은 응용프로그람제품에 대하여 또 다른 하나의 실험을 진행하고 발견된 모든 오유의 87%가 설계와 코드검토에서 발견되였다는것을 알아 내였다[Fagan, 1976]. 검토의 한가지 유용한 측면효과는 모듈시험에 적은 시간이 들여 지기때문에 프로그람작성자의 능률이 올라 가게 되였다는것이다. 자동화된 평가모형을 리용하여 파간은 검토결과 검토에 시간이 걸렸음에도 불구하고 프로그람작성자의 자원의 25%가 절약되였다고 결정하였다.

다른 실험에서 존스는 발견된 오유의 70%이상이 설계와 코드검토를 진행하는 과정에 발견되였다는것을 밝히였다[Jones, 1978].

이후의 연구들에서 마찬가지로 흥미 있는 결과들이 얻어 졌다. 6,000행의 업무자료처리응용프로그람에서 발견된 모든 오유의 93%는 검토기간에 발견되였다[Fagan, 1986]. 문헌 [Ackerman, Buchwald, and Lewski, 1989]에서 보고된바와 같이 조작체계의 개발기간에 시험이 아니라 검토를 리용하는것은 오유를 발견하는데 드는 비용을 85%까지 감소시켰다. 례하면 절환체계제품에서는 90%감소되였다[Fowler, 1986]. 분사식발동기연구소(JPL)에서는 평균 매 사람이 2h 진행한 검토에서 4개의 주요한 오유와 14개의 작은 오유가 발견되였다[Bush, 1990]. 딸라로 바꾸어 환산하면 이것은 대략 한 검토당 2만 5천딸라를 절약한것으로 된다. 또한 다른 JPL연구[Kelly, Sherif, and Hops, 1992]에서는 발견된 오유의 개수가 단계마다 지수함수적으로 감소한다는것을 보여 주었다. 달리 말하면 검토수단을 리용하여 쏘프트웨어개발공정에서 신속하게 오유들을 발견할수 있었다. 이러한 신속오유발견의 중요성은 그림 1-5에 반영되여 있다.

검토공정에서 위험요소는 그것이 관통심사회의와 마찬가지로 성능평가를 위하여 리용될수 있다는데 있다. 리용가능한 세부오유정보로 인하여 검토의 경우에 위험은 특히 심각하게 제기된다. 파간은 3년간에 걸치는 고찰을 통하여 자기가 그 어떤 IBM관리자도 프로그람작성자에 대하여 이런 정보를 리용하지 않으며 또는 그가 주장하는바와 같이 《목전의 리익을 위하여 장래의 리익을 희생시키》려고 하는 관리자도 없다는것을 알았다고 서술하는것으로써 이러한 우려를 가셔 냈다[Fagan, 1976]. 그러나 검토가 철저히 진행되지 않으면 IBM에서와 같은 그러한 큰 성과를 거둘수 없다. 만일 최고관리자측이 잠재적문제를 알아 차리지 못하는 한 검토정보를 잘못 리용할 가능성이 명백히 존재한다.

6. 2. 4. 검토와 관통심사회의의 비교

얼핏 보기에 검토와 관통심사회의사이의 차이는 검토팀이 오유를 찾는데 도움을 주는 질문에 대한 검열목록을 리용한다는것이다. 그러나 차이는 그보다 더 심오하다. 관통심사회의는 두 단계로 진행되는 과정이다. 즉 준비와 그것에 뒤이은 문서에 대한 분석이다. 검토는 5단계의 과정 즉 개괄, 준비, 검토, 재작업, 뒤조사로 이루어 지며 이 매 단계에서 지켜야 할 절차들은 형식화된다. 이와 같은 형식화의 실례는 오유들을 주의깊게 분류하고 그 정보를 장래의 제품검토에서와 마찬가지로 련속되는 단계들에서의 문서에 대한 검토에 리용하는것이다. 검토과정은 관통심사회의보다 시간이 훨씬 더 오래 걸린다. 검토가 추가적인 시간과 노력을 보상할수 있는가? 선행한 절에서의 자료들은 검토가 오유를 발견하기 위한 강력하고 비용상 효과적인 도구이라는것을 명백히 지적하고 있다.

6. 2. 5. 심사의 우결함

심사(관통심사회의나 검토)는 두개의 주요한 우점이 있다. 첫째로, 심사는 오유를 발견하는 효과적인 방법이라는것이며 둘째로, 오유가 쏘프트웨어개발공정의 조기단계에서

즉 오유를 수정하는데 비용이 들기전에 발견된다는것이다. 실례로 설계오유는 실행단계가 개시되기전에 발견되고 코드작성오유는 모둘이 제품으로 통합되기전에 발견된다.

그러나 쏘프트웨어개발공정이 적당치 못하면 심사의 효과는 감소될수 있다. 첫째로, 대규모쏘프트웨어는 보다 작고 독립적인 구성요소들로 이루어 져 있지 않는 한에서는 심사하기가 아주 곤난하다. 객체지향파라다임의 우점의 하나는 정확히 수행되는 경우에 결과적인 제품이 실지로 독립적인 부분들로 구성된다는것이다. 둘째로, 설계심사팀은 때때로 명세서를 참고해야 한다. 즉 코드심사팀은 흔히 설계문서에 자주 접근할 필요가 있다. 만일 선행단계의 문서작성이 완성되지 않고 또한 프로젝트의 현재 판본을 반영하도록 갱신되지 않으며 직결식으로 리용할수 없는한 심사팀의 효과성은 심히 제한된다.

6．2．6．검토를 위한 척도

검토의 효과성을 결정하기 위하여 여러가지 각이한 척도가 리용될수 있다. 첫째가 오유밀도(*fault density*)이다. 명세서와 설계가 검토될 때 검토된 페지당 오유개수가 측정될수 있다. 즉 코드검토에 알맞는 척도는 검토된 코드 1,000행당 오유개수(KLOC)이다. 이러한 척도를 자료단위당 중요한 오유와 자료단위당 부차적인 오유로 가를수 있다. 또 하나의 유용한 척도는 오유검출률(*fault detection rate*)이다. 즉 시간당 발견된 중요한 오유와 부차적인 오유의 개수이다. 세번째 척도는 오유검출효률(*fault detection efficiency*) 즉 한사람이 한시간동안 발견한 중요한 오유와 부차적인 오유의 개수이다.

비록 이러한 척도들의 목적이 검토과정에 대한 효과성을 측정하는데 있더라도 반대로 결과는 개발팀의 결함을 반영할수도 있다. 실례로 만일 오유검출률이 갑자기 20KLOC로부터 30KLOC로 증가하였다면 이것은 검토팀이 갑자기 50%정도 더 효과적으로 일했다는것을 의미하지는 않는다. 이것에 대한 다른 하나의 설명은 코드의 품질이 감소되였고 발견된 오유가 더 많아 졌다는것이다.

비실행에 대한 시험을 론의하였으므로 다음의 문제점은 실행에 기초한 시험이다.

6．3．실행에 기초한 시험

시험은 오유가 존재하지 않는다는것을 보여 주는것이라고 주장되여 왔다. 오유(*fault*)는 일반적으로 바그(*bug*)라고 하는것에 대한 IEEE규격용어이며 반면에 고장(*failure*)은 오유의 결과로 하여 제품에 대하여 관측된 부정확한 동작이다[IEEE 610.12, 1990]. 마지막으로 착오(*error*)는 프로그람작성자에 의해서 이루어 진 잘못을 의미한다.). 지어 일부 기업체들이 쏘프트웨어예산의 50%를 시험에 돌린다고 해도 배포되는 《시험된》 제품은 전적으로 믿을수는 없다.

이러한 모순이 있게 되는 리유는 단순하다. 디지크스트라가 제기한바와 같이 《프로그람시험은 바그의 존재를 보여 주는 아주 효과적인 방법으로 될수 있다. 그러나 그것은 바그가 존재하지 않는다는것을 보여 주는데는 기대할수 없을 정도로 부적당하다.》[Dijkstra, 1972]. 디지크스트라가 말하고 있는것은 만일 어떤 제품이 시험실례를 리용하여

실행되고 그 출력이 잘못되였다면 그 제품은 결정적으로 어떤 오유를 포함하고 있다는것이다. 그러나 출력이 정확하다고 해도 제품에는 여러가지 오유가 있을수 있다. 즉 특정한 시험으로부터 도출될수 있는 유일한 정보는 그 제품이 특정한 시험실례의 모임에 대하여 정확히 동작하고 있다는것이다.

6. 4. 무엇이 시험되여야 하는가

어떤 특성이 시험되여야 한다는것을 설명하려면 먼저 실행에 기초한 시험에 대하여 정확한 설명을 주어야 한다.

구데나흐에 의하면 실행에 기초한 시험은 선택된 입력을 리용하여 이미 알려 진 환경에서 부분적으로 제품이 실행된 결과에 기초하여 제품의 명백한 동작특성을 추론하는 과정이다[Goodenough, 1979]. 이 정의에는 세가지 곤난한 의미가 포함되여 있다. 첫째로 그 정의는 시험이 추론과정이라고 서술하고 있다. 시험자는 제품을 이미 알려 진 입력자료를 리용하여 동작시킨 다음 출력을 시험한다. 시험자는 제품에 그 무엇인가 잘못된것이 있는가를 추론해야 한다. 이러한 관점으로부터 시험은 널리 알려 져 있는 어두운 방에서의 검은고양이찾기와 비교할수 있다. 시험자는 어떤 오유를 찾는데 도움이 되는 실마리를 거의나 가지고 있지 못하다. 즉 10～20개의 입력자료와 대응하는 출력자료의 모임에 그리고 혹시 사용자가 작성한 오유보고서, 수행되는 코드만이 있을뿐이다. 이로부터 시험자는 오유가 있는가, 있으면 오유가 무엇인가 하는것를 추론해야 한다.

정의에서의 두번째 문제는 《이미 알려 진 환경》이라는 표현으로부터 발생한다. 우리는 사실상 하드웨어라든가 쏘프트웨어와 같은 우리의 환경에 대하여 전혀 알수 없다. 우리는 절대로 조작체계가 정확히 기능을 수행하고 있다거나 혹은 실시간루틴들이 정확하다는것을 확정할수 없다. 콤퓨터의 주기억에서 때때로 장치적인 고장이 있을수 있다. 그래서 사실상 제품의 거동으로서 관측되는것은 오유콤파일러, 오유하드웨어, 환경에서의 어떤 다른 환경요소와 호상작용하는 정확한 제품일수 있다.

실행에 기초한 시험에 대한 정의의 세번째 문제는 《선택된 입력을 리용하여》라는 표현이다. 실시간체계의 경우에는 흔히 체계의 입력에 대하여 그 어떤 조종도 할수 없다. 항공우주학부문의 쏘프트웨어를 생각해 보자. 비행조종체계는 두가지 류형의 입력을 가진다. 첫번째 류형의 입력은 조종사가 비행기에 대하여 진행하려고 하는 동작이다. 결국 만일 조종사가 올라 가려고 조종간을 뒤로 당긴다거나 혹은 비행기의 속도를 높이기 위하여 조절변을 연다고 하면 이런 기계적인 움직임은 수자신호로 변환되여 비행조종콤퓨터에 보내여 진다. 두번째 류형의 입력은 비행기의 고도라든가 속도 그리고 바람에 의해서 날개가 흔들리는 정도 등과 같은 현재의 물리적인 상태들이다. 비행조종쏘프트웨어는 이러한 량들에 대한 값들을 리용하여 날개라든가 기관과 같은 비행기의 구성부분들에 어떤 신호를 보내겠는가를 계산하여 지령을 실현한다. 조종사의 입력은 비행기의 조종에 알맞게 설정됨으로써 쉽게 간단히 요구하는 값들로 설정될수 있는 반면에 비행기의 현재의 물리적인 상태에 대응한 입력은 그렇게 쉽게 조작될수는 없다. 사실상 비행기에 《선택된 입력》를 제공할수 있는 방도는 없다. 그러면 이러한 실시간체계는 어떻게 시

험될수 있는가? 대답은 모의기를 리용하는것이다.

모의기(*simulator*)는 비행조종쏘프트웨어의 경우에 제품이 실행되는 환경에 대한 작업모형이다. 비행조종쏘프트웨어는 모의기가 비행조종쏘프트웨어에 선택된 입력을 보내도록 함으로써 시험될수 있다. 모의기는 조작자가 입력변수를 그 어떤 선택된 값으로 설정할수 있도록 조종을 진행한다. 만일 시험의 목적이 실제적인 비행기기관에 불이 달렸을 때 비행조종쏘프트웨어가 어떻게 동작하여야 하는가를 결정하는것이라면 모의기의 조종이 설정되였다고 하더라도 비행조종쏘프트웨어에 보내여 진 입력은 실제비행기의 기관이 다 타버린 경우에 있을수 있는 입력과 구별할수 없게 된다. 출력에 대해서는 비행조종쏘프트웨어로부터 모의기에 보내여 진 출력신호를 시험함으로써 분석이 진행된다. 그러나 모의기는 기껏하여 체계의 어떤 측면의 실제모형에 대한 훌륭한 근사로 될수 있는것이다. 즉 모의기는 절대로 체계 그자체로는 될수 없다. 모의기를 리용하는것은 《알려 진 환경》이 사실상 있는 반면에 이러한 알려 진 환경이 모든 면에서 그 제품이 설치될 실제환경과 같다고는 거의나 볼수 없다는것을 의미한다(실시간쏘프트웨어를 시험하는 문제를 파악하기 위해서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

실시간쏘프트웨어는 흔히 대부분의 사람들 즉 그 개발자들조차도 리해하기 복잡하다. 결과 매우 기능이 높은 시험자들도 일반적으로 발견할수 없는 요소들사이에서 파악하기 힘든 호상작용이 발생할수 있으며 따라서 걸보기에는 중요치 않은 변화들이 아주 중요한 결과를 초래할수도 있다. 이에 대한 유명한 실례는 1981년 4월에 첫 우주왕복선의 궤도비행이 지연되게 된 오유를 들수 있다[Garman 1981]. 우주왕복선은 4개의 동일한 동기화된 콤퓨터에 의하여 조종된다. 또한 하나의 독립적인 다섯번째 콤퓨터가 4개의 콤퓨터가 고장날 경우를 고려하여 예비로 준비된다. 2년전에 우주왕복선의 콤퓨터들이 동기화되기전에 모선초기화를 진행하는 모듈에 변경이 가해 졌다. 이러한 변경에서 하나의 유감스러운 효과는 현재시간보다 약간 늦은 시간을 포함하고 있는 기록을 우주왕복선의 콤퓨터들의 동기화를 위하여 리용되는 자료령역에 잘못 보낸것이다. 이 시간은 이러한 오유가 발견되지 않을 정도로 실제시간에 매우 가까왔다. 약 1년후에 시간차이가 약간 커지게 되고 67번중 한번의 오유를 발생시킬수 있는 정도로 되였다. 그다음 첫 우주왕복비행이 진행되는 날에 동기화실패가 발생하였으며 4개의 콤퓨터들중 3개가 첫 콤퓨터보다 한주기 늦게 동기화되였다. 만일 4대의 콤퓨터가 일치하지 않으면 이 콤퓨터들로부터 오는 정보를 다섯번째 콤퓨터가 접수하지 않도록 하는 장애안전장치가 다섯번째 콤퓨터의 동기화를 가로막는 예상치 않는 결과를 발생시켰으며 이로 하여 우주비행은 지연되였다. 이 우발적인 사건의 그처럼 유명하게 된것은 오유가 모선초기화모듈에 있었다는것이다. 즉 하나의 모듈이 명백히 동기화루틴과 련결되지 않았던것이다.

시험에 대한 앞의 정의에서는 《동작특성》에 대하여 서술하고 있다. 어떤 동작특성이 시험되는가? 그에 대한 명백한 대답은 제품이 정확하게 기능하는가에 대한 시험이라는것이다. 그러나 보여 준바와 같이 정확성은 필요조건도 충분조건도 아니다. 정확성에

대하여 론의하기전에 4개의 기타 다른 동작특성 즉 유용성과 믿음성, 로바스트성 그리고 성능을 고찰해야 한다[Goodenough, 1979].

6．4．1．유용성

유용성(*utility*)은 정확한 제품이 그 명세서에 허용된 조건하에서 리용될 때 사용자들의 요구가 만족되는 정도를 의미한다. 달리 말하면 정확히 기능을 수행하는 제품은 명세서에 의해서 타당한 입력에 복종한다. 사용자들은 다음과 같은것들을 시험할수 있다. 즉 제품을 리용하기 쉬운가, 제품이 유용한 기능을 수행하는가, 제품이 경쟁하고 있는 제품들에 비해서 비용상 효과적인가를 시험할수 있다. 제품이 정확성여부에 관계없이 이와같이 중요한 문제들은 시험되여야 한다. 만일 제품이 비용상 효과적이지 못하면 그것을 구입할 때 리익이 없다. 만일 제품을 리용하기 힘들면 그것은 전혀 리용되지 않거나 혹은 부정확하게 리용된다. 따라서 현존제품(수축포장된 쏘프트웨어를 포함하여)의 구입을 고려할 때 제품의 유용성을 우선적으로 시험하고 제품이 이런 점에서 만족되지 않으면 시험을 중지한다.

6．4．2．믿음성

제품에 대하여 시험해야 할 또 다른 한 측면은 믿음성이다.

믿음성(*reliability*)은 제품실패의 빈도와 위험성에 대한 측정값이다. 즉 실패는 허용되는 조작조건하에서 오유에 의해서 초래되는 받아 들일수 없는 효과나 동작이라는것을 상기하자. 달리 말하면 제품이 얼마나 자주 실패하는가(실패사이의 평균시간)와 그 실패의 결과가 얼마나 나쁜가 하는것을 알아야 한다. 제품이 실패될 때 중요하게 론의해야 할 문제는 그것을 퇴치하는데 평균 얼마만한 시간이 걸리는가 하는것이다. 그러나 흔히 실패한 결과를 퇴치하는 시간이 얼마나 걸리는가 하는것이 더 중요하다. 이 마지막문제점은 흔히 스쳐 버릴수 있다. 통신앞단에서 실행되고 있는 쏘프트웨어가 평균 6개월에 한번씩 실패한다고 가정하자. 그러나 그것이 실패하면 그것은 자료기지를 완전히 없애 버린다. 기껏하여 자료기지는 마지막검사점이 얻어 졌을 때 자기 상태로 다시 초기화될수 있으며 검사흔적은 자료기지를 그것이 갱신된 상태로 두는데 리용될수 있다. 그러나 만일 이러한 회복공정이 자료기지와 통신앞단이 동작하지 않는 2일동안에 보다 좋은 역할을 수행한다면 실패사이의 평균시간이 6개월로 되는데도 불구하고 제품의 믿음성은 낮아진다.

6．4．3．로바스트성

매 제품에 대하여 시험되여야 할 또 다른 하나의 측면은 로바스트성이다. 정확히 정의하기는 힘들지만 로바스트성(*robustness*)은 본질에 있어서 동작조건의 범위라든가 타당한 입력에 대한 접수할수 없는 결과가 발생할 가능성 그리고 제품에 입력되는 비타당한 입력에 대한 허용가능성과 같은 여러가지 요인들에 기인한다. 허용동작조건이 넓은 제품

은 보다 제한된 제품보다 로바스트성이 강하다고 말할수 있다. 로바스트적인 제품은 입력이 명세서를 만족하는 경우에는 접수불가능한 결과를 산생시키지 말아야 한다. 실례로 타당한 지령을 주는것은 해로운 결과를 초래하지 말아야 한다. 로바스트적인 제품은 그 제품이 허용동작조건하에서 리용되지 않을 때에도 폭주되지 말아야 한다. 로바스트성의 측면을 시험하기 위하여 명세서의 입력조건을 만족시키지 않는 시험실례를 일부러 넣어 주고 시험자는 그 제품이 얼마나 나쁘게 작용하는가를 결정한다. 실례로 제품이 이름을 요청했을 때 시험자는 Control-A escape-%? $#@와 같은 허용되지 않는 문자렬로 응답할 수 있다. 만일 콤퓨터가 Incorrect data-Try again과 같은 통보문을 내보내거나 더 좋기는 자료가 예정된 규칙을 준수하지 않고 있는 리유를 사용자에게 알려 준다면 그것은 자료가 요구한것보다 약간 차이나도 폭주되는 제품보다는 로바스트성이 더 강할것이다.

6. 4. 4. 성 능

성능(*performance*)은 시험되여야 할 제품의 또 하나의 측면이다. 실례로 제품이 응답시간이나 기억공간요구와 관련한 제약을 만족하는 정도를 알아야 한다. 반항공유도미싸일에 있는 콤퓨터와 같은 내장형콤퓨터체계에 대한 기억공간제약은 주기억의 128kB만이 쏘프트웨어에 리용될수 있다는것과 같은것일수 있다. 쏘프트웨어가 아무리 우월하다고 해도 만일 256kB의 주기억을 요구한다면 그때는 그 쏘프트웨어를 전혀 리용할수 없다(내장형쏘프트웨어에 대하여 더 알기 위해서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

내장형콤퓨터는 계산을 기본목적으로 하지 않는 큰 체계의 필수적인 부분이다. 내장형쏘프트웨어의 기능은 콤퓨터가 내장되여 있는 장치들을 조종하는것이다. 군사분야의 실례를 들어 보면 군용비행기안에 설치된 항공용콤퓨터나 대륙간탄도미싸일에 설치된 콤퓨터들의 망이 있다. 미싸일의 원추형머리부에 있는 내장형콤퓨터는 그 미싸일만을 조종하며 미싸일기지에 있는 병사들의 로임지불표를 인쇄하는데는 리용될수 없다. 보다 잘 알려진 실례는 수자식시계나 세탁기에 있는 콤퓨터소편이다. 세탁기에 있는 소편은 전문적으로 세탁기를 조종하는데 리용된다. 그 세탁기의 주인은 장부를 결산하는데는 소편을 쓸수가 없다.

실시간쏘프트웨어는 장치시간제약에 대하여 특징 지어 진다. 즉 시간제약은 제약을 만족시키지 않으면 정보를 잃을수 있다는 특성을 가진다. 실례로 핵반응로조종체계는 핵의 온도를 표본화하고 10s마다 그 자료를 처리해야 한다. 만일 체계가 매 10s마다 온도수감장치로부터 중단신호를 조종할수 있을 정도로 충분히 빠르지 못하다면 그때는 자료를 잃어 버리며 자료를 되찾을수 없다. 즉 체계가 온도자료를 받아 들이게 되는 다음번 시각에는 잃어 버린 자료가 아니라 현재의 온도를 읽게 된다. 만일 반응로가 녹음점상에 있으면 명세서에 규정된것처럼 련관된 모든 정보들이 수신되고 처리되는것이 중요하다. 모든 실시간체계들에서 성능은 명세서에 지적된 모든 시간제약을 만족시켜야 한다.

6. 4. 5. 정확성

마지막으로 정확성에 대한 정의가 주어 질수 있다. 만일 제품이 허용되는 조건에서 동작될 때 계산자원리용에 독립인 출력명세서를 만족시키면 제품은 정확하다고 한다 [Goodenough, 1979]. 달리 말하면 입력명세서를 만족시키는 입력이 주어 지고 제품에 필요한 자원이 모두 주어 지면 제품은 출력이 출력명세서를 만족시킨다면 정확하다.

시험 그자체에 대한 정의에서와 같이 정확성(*correctness*)에 대한 이 정의는 의미적으로 우려를 자아내고 있다. 제품이 아주 다양한 시험실례에 대하여 성공적으로 시험되였다고 하자. 이것은 이 제품을 접수할수 있다는것을 의미하는가? 유감스럽게도 그렇지는 않다. 만일 제품이 정확하다면 그것은 제품이 명세서를 만족시킨다는것을 의미한다. 그러나 명세서 그자체가 부정확하다면 어떻게 되겠는가? 이러한 난점을 례증하기 위하여 그림 6-1에서 보여 준 명세서를 고찰하자. 명세서는 정렬에 대한 입력이 n개의 옹근수들의 배렬 p인 반면에 출력은 비감소순서로 정렬된 또 다른 하나의 배렬 q이라는것을 서술하고 있다. 얼핏 보기에는 명세서가 완전히 정확한것처럼 보인다. 그러나 그림 6-2에서 보여 준 방법 tricksort를 고찰해 보자. 이 방법에서 배렬 q의 n개요소들은 모두 0으로 설정되여 있다. 이 방법은 그림 6-1의 명세서를 만족시키며 따라서 정확하다.

입력명세서:	p : n개의 옹근수의 배렬, n>0
출력명세서:	q : q[0]≦ q[1]≦ …≦q[n-1]인 n개의 옹근수의 배렬

그림 6-1. 정렬에 대한 부정확한 명세서

```
void trickSort (int p[], int q[])
{
        int i;
        for ( i=0; I<n; i++)
            q[i]=0;
}
```

그림 6-2. 그림 6-1의 명세서를 만족시키는 방법 trickSort

무슨 일이 일어 나겠는가? 유감스럽게도 그림 6-1의 명세서는 옳지 않다. 출력배렬 q의 원소들이 입력배렬 p의 원소들에 대한 재정렬이라는 서술문이 생략되였다. 정렬에서 본질적인 측면은 그것이 재배치과정이라는것이다. 그리고 그림 6-2의 방법은 명세서의 이러한 요구를 반영하고 있다. 달리 말하면 방법 trickSort는 정확하지만 그림 6-1의 명세서는 옳지 않다. 정정된 명세서를 그림 6-3에 보여 준다.

입력명세서: p : n개의 옹근수의 배렬, n>0
출력명세서: q : q[0]≦ q[1]≦ …≦q[n-1]인 n개의 옹근수의 배렬
배렬 q의 요소들은 변경되지 않은 배렬 q의
요소들의 교체이다

그림 6-3. 정렬에 대한 정정된 명세서

우의 실례로부터 명백히 명세서오유의 결과는 중요하다. 결국 명세서가 부정확하면 제품의 정확성에 대하여 말할수 없다.

어떤 제품이 정확하다고 하는 사실은 충분하지 않다. 왜냐하면 정확하다고 보여 준 명세서 그자체가 옳지 않을수 있기때문이다. 그러나 그것이 필요한가? 다음의 실례를 고찰해 보자. 어느 한 쏘프트웨어기업체가 새로운 C++콤파일러를 얻게 되였다. 새로운 콤파일러는 낡은 콤파일러에 비해서 1min동안에 2배나 많은 행을 가진 원천코드를 번역할수 있으며 목적코드를 거의나 45% 더 빨리 실행할수 있으며 목적코드의 규모는 약 20% 더 작다. 이밖에 오유통보문은 훨씬 더 명백하고 년간 유지정비와 갱신에 드는 비용은 낡은 콤파일러에서의 절반도 안된다. 그러나 여기에 하나의 문제점이 있다. 즉 임의의 클라스에서 **for**명령문이 나타난 첫 시각에 콤파일러는 거짓오유통보문을 인쇄한다. 따라서 콤파일러는 정확치 않다. 왜냐하면 콤파일러에 대한 명세서는 원천코드에 오유가 있을때에만 암시적으로 혹은 명백히 오유통보문을 인쇄할것을 요구하고 있기때문이다. 콤파일러는 확실히 리용할수는 있다. 즉 사실상 한 경우를 제외한 모든 방법에서 콤파일러가 절대적으로 리상적이다. 더우기 이러한 부차적인 오유가 그다음의 해제과정에서 정확할것이라는것을 기대하는것은 아주 타당이다. 그러는 과정에 프로그람작성자는 거짓오유통보문을 무시하는것을 배우게 된다. 기업체는 부정확한 콤파일러를 가지고 일할수 있을뿐아니라 만일 누군가가 그것을 보다 낡은 정확한 콤파일러로 교체할것을 제기하면 항의할것이다. 따라서 제품의 정확성은 필요조건도 충분조건도 아니다.

우의 두 실례는 약간 인위적이다. 그러나 그것들은 정확성이 단순히 제품이 명세서에 대한 하나의 정확한 실현이라는것을 의미하고 있다. 달리 말하면 제품이 정확하다는것을 보여 주는것보다도 시험하는것이 더 낫다. 실행에 기초한 시험과 련관된 모든 난점들에 대처하여 콤퓨터과학자들은 어떤 제품이 자기의 사명을 수행하고 있다는것을 담보해주는 다른 방법을 제기하기 위하여 시도하였다. 40여년이상 상당한 주의를 끌어 온 이와 같은 한가지 비실행에 기초한 시험은 정확성증명이다.

6. 5. 시험과 정확성증명의 대비

정확성증명은 제품이 정확하다는것 즉 달리 말하면 명세서를 만족시킨다는것을 보여주는 수학적기법이다. 때때로 이 기법을 검증이라고 부른다. 그러나 이미 강조한바와 같이 용어 검증은 흔히 정확성증명뿐아니라 모든 비실행에 기초한 기법들을 의미하는데 리용된다. 명백히 말하면 수학적인 절차는 정확성증명이라고 불리울것이다. 독자들에게 그

것이 수학적인 증명과정이라는것을 상기시키기 위하여 정확성증명(*correctness proving*)이라고 한다.

6. 5. 1. 정확성증명의 실례

정확성을 어떻게 증명하는가를 보기 위하여 그림 6-4에서 보여 준 코드토막을 고찰해 보자. 코드와 등가인 흐름도표를 그림 6-5에 보여 준다. 이제 우리는 코드토막이 정확하다는것을 보게 된다. 즉 코드가 실행된 다음에 변수 s는 배렬 y의 n개의 원소들의 합을 포함할것이라는것을 보자. 그림 6-6에서는 하나의 단언(*assertion*)이 매 명령문의 앞뒤에 즉 문자 *A*부터 *H*로 표식된 위치에 놓인다. 즉 어떤 수학적성질이 성립하는 매 위치에서 하나의 주장을 주었다.

이제 이러한 매 주장의 정확성을 증명한다.

입력명세서 즉 코드가 실행되기전에 *A*에서 성립하는 조건은 변수 n이 옹근수이라는것이다. 즉

$$\boldsymbol{A}\text{:}\quad n \in \{1,\ 2,\ 3,\ \cdots\} \tag{6.1}$$

명백한 출력명세서는 만일 조종이 *H*에 이르면 변수 s의 값이 *y*에 기억된 n개의 값들의 합이라는것이다. 즉

$$\boldsymbol{H}\text{:}\quad s = y[0] + y[1] + \cdots + y[n-1] \tag{6.2}$$

사실상 코드토막은 보다 강한 출력명세서에 관하여 정확하다는것이 증명될수 있다. 즉

$$\boldsymbol{H}\text{:}\quad k = n \quad \text{and} \quad s = y[0] + y[1] + \cdots + y[n-1] \tag{6.3}$$

```
int k,s;
int y[n];
k=0;
s=0;
while ( k<n)
{
    s=s+y[k];
    k=k+1;
}
```

그림 6-4. 정확성이 증명된 코드부분

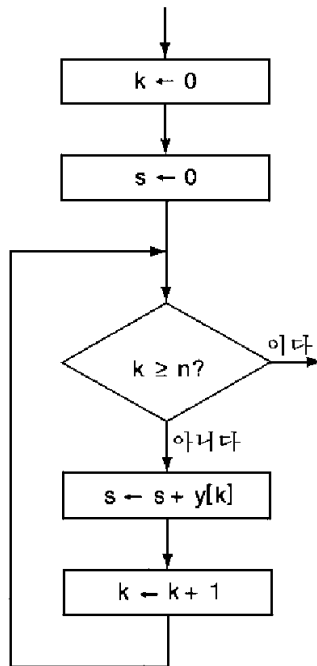



그림 6-5. 그림 6-4의 흐름도

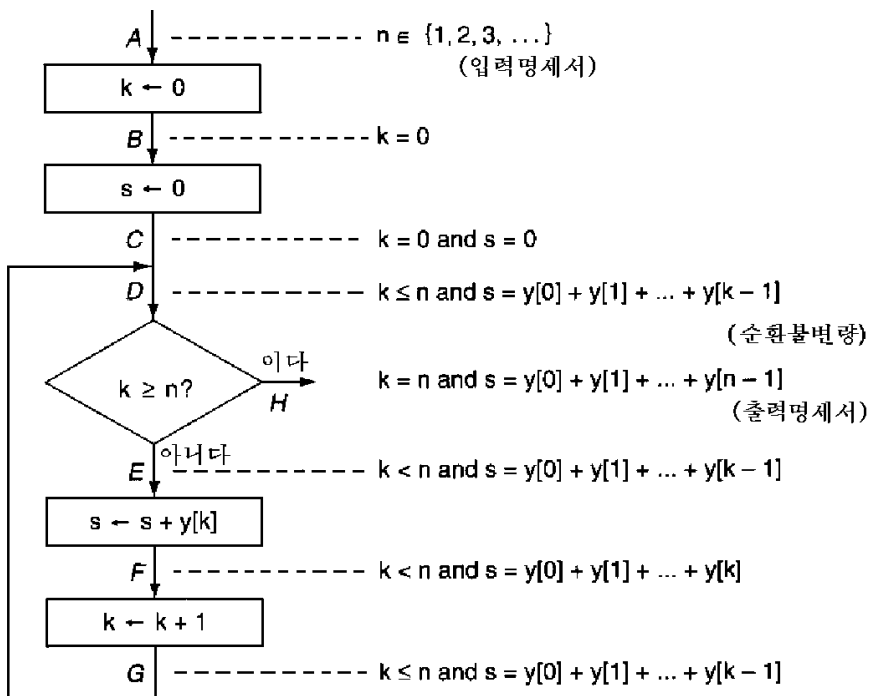



그림 6-6. 입력명세서, 출력명세서, 순환불변량, 단언을 포함한 그림 6-5

마지막명령문에 대한 본래의 작용은 출력명세서(6.3)가 어디에서부터 오는가를 물어보는것이다. 증명의 마감에서 그 질문에 대답할것을 요구한다. 즉 문제 6.10과 6.11을 보시오. 입력명세서와 출력명세서외에 증명과정의 세번째 측면은 순환을 위한 불변량을 제공하는것이다. 즉 하나의 수학적표현식이 순환이 0.1 또는 여러번 실행되였는가 아닌가에 관계없이 점 D에서 성립한다는것을 증명하여야 한다. 성립한다고 증명될 순환불변량은 다음과 같다.

$$\boldsymbol{D}: k \leqq n \text{ and } s = y[0] + y[1] + \cdots + y[n-1] \tag{6.4}$$

이제 만일 입력명세서(6.1)가 점 *A*에서 성립한다면 출력명세서(6.3)는 점 *H*에서 성립한다는것을 보여 준다. 즉 코드토막이 정확하다고 증명된다.

처음에 값주기명령문 $k \leftarrow 0$ 이 실행된다. 이제는 조종이 점 *B*에 있게 되는데 여기서 다음의 단언이 성립한다. 즉

$$\boldsymbol{B}: \quad k = 0 \tag{6.5}$$

더 정확하게 점 *B*에서 단언은 $k=0$ and $n \in \{1, 1, 3, \cdots\}$을 읽어야 한다. 그러나 입력명세서(6.1)는 흐름도표의 모든 점에서 성립한다. 그러므로 and $n \in \{1, 2, 3, \cdots\}$은 지금부터는 생략된다.

점 C에서 두번째 값주기명령문 $s \leftarrow 0$ 로부터 다음의 단언이 성립한다.

$$\boldsymbol{C}: \quad k=0 \text{ and } s=0 \tag{6.6}$$

이제는 순환이 입력된다. 순환불변량(6.4)이 정말 정확하다는것이 귀납적으로 증명될것이다. 순환이 처음으로 실행되기전에 단언(6.6)이 성립한다. 즉 k=0이며 s=0이다. 이제 순환불변량(6.4)을 고찰하자. 단언(6.6)에 의해서 k=0이고 입력명세서(6.1)로부터 $n \geqq 1$이기때문에 요구한바와 같이 $k \leq n$ 이라는것이 나온다. 더우기 k=0이기때문에 $k-1= -1$로 되며 따라서 (6.4)에서의 합은 요구한바와 같이 비게 되고 s=0으로 된다. 그렇기때문에 순환불변량 (6.4)은 첫 순환이 입력되기전에 참으로 된다.

이제부터는 귀납의 가정단계가 진행된다. 코드토막이 실행되고 있는 어떤 단계에서 순환불변량이 성립한다고 가정하자. 즉 어떤 값 k_0, $0 \leqq k_0 \leqq n$와 같은 k에 대하여 실행은 점 *D*에 있으며 다음의 단언이 성립한다.

$$\boldsymbol{D}: k_0 \leqq n \text{ and } s= y[0] + y[1] + \cdots + y[k_0-1] \tag{6.7}$$

이제 조종이 조건시험통을 통과한다. 만일 $k_0 \geqq n$이면 가정에 의하여 $k_0 \leqq n$이기때문에 k0=n으로 된다. 귀납의 가정 (6.7)에 의해서 이것은 다음의 사실을 의미한다.

$$\boldsymbol{H}: k_0 \leqq n \text{ and } s= y[0] + y[1] + \cdots + y[n-1] \tag{6.8}$$

이것은 정확히 출력명세서(6.3)이다.

다른 한편 만일 조건검사 $k_0 \geqq n$? 이 실패하면 그때 조종은 점 *D*에서 점 *E*로 넘어간다. k_0은 n보다 같거나 크지 않기때문에 $k_0<0$이며 (6.7)은 다음과 같이 된다.

$$E: k_0<n \text{ and } s= y[0] + y[1] + \cdots + y[k_0-1] \tag{6.9}$$

이제 명령문 s ← s + $y[k_0]$이 실현되고 따라서 F에 있는 단언 (6.9)로부터 다음의 단언이 성립한다.

$$F: k_0<n \text{ and } s = y[0] + y[1] + \cdots + y[k_0-1] + y[k_0] \\ = y[0] + y[1] + \cdots + y[k_0] \tag{6.10}$$

다음에 실행되게 되는 명령문은 $k_0 \leftarrow k_0 + 1$이다. 이 명령문의 효과를 보기 위하여 이 명령문이 실행되기전 k_0값이 17이라고 가정하자. 그러면 (6.10)에 있는 더하기식에서 마지막항은 y[17]로 된다. 이제 값 k_0이 하나씩 증가되여 18로 된다. 합 s는 변화되지 않는다. 따라서 더하기식에서 마지막항은 여전히 y[17]로 되는데 이것은 지금 $y[k_0-1]$로 되여 있다. 또한 점 F에서는 $k_0<n$이다. 값 k_0이 하나씩 증가한다는것은 부등식이 점 *G*에서 성립하기로 되여 있다면 그때 $k_0 \leqq n$이라는것을 의미한다. 결국 k_0이 하나씩 증가한 결과 점 *G*에서 다음과 같은 단언이 성립한다. 즉

$$G: k_0<n \text{ and } s = y[0] + y[1] + \cdots + y[k_0-1] \tag{6.11}$$

점 *G*에서 성립하는 단언 (6.11)은 가정에 의하여 점 *D*에서 성립하는 단언 (6.7)과 같다. 그러나 점 *D*는 점 *G*와 위상기하학적으로 동등하다. 달리 말하면 (6.7)이 $k = k_0$에 대해서 점 *D*에서 성립한다면 다시 점 *D*에서 $k = k_0 + 1$이 성립한다. 순환이 k = 0에 대하여 성립한다는것은 이미 보여 주었다. 귀납에 의해서 순환불변량(6.4)을 모든 k, $0 \leqq k \leqq n$에 대하여 성립하는것으로 된다.

이제 남은것은 순환종결을 증명하는것이다. 처음에 단언(6.6)에 의하여 k의 값은 0과 같다. 순환이 매번 반복되면 명령문 k ← k + 1이 실행될 때마다 k값은 하나씩 증가한다. 결국 k는 값 n에 도달하여야 하며 이때 순환은 끝나며 s의 값은 단언(6.8)에 의해서 계산되고 따라서 출력명세서(6.3)를 만족하게 된다.

검토를 위하여 입력명세서(6.1)가 주어 졌을 때 순환불변량(6.4)이 순환이 0.1 또는 그이상 실행될 때에 성립한다는것이 증명되였다. 더우기 n번 반복한 다음순환이 끝나고 그때 k와 s의 값들은 출력명세서(6.3)를 만족시킨다는것이 증명되였다. 달리 말하면 그림 6-4의 코드토막은 수학적으로 정확하다는것이 증명되였다.

6. 5. 2. 정확성증명의 실례연구

정확성증명의 중요한 측면은 그것이 설계 및 코드작성과 결합하여 진행된다는것이다. 디지크스트라가 제기한바와 같이 《프로그람작성자는 프로그람을 증명하게 되며 프로그람을 협력하여 완성하여야 한다.》[Dijkstra, 1972]. 실례로 순환이 설계에 병합될 때 순환불변량이 제시된다. 즉 설계가 계단식으로 세련되기때문에 불변량이 있게 된다. 이런 방법으로 제품을 개발하는것은 프로그람작성자에게 제품이 정확하다는 믿음을 주게 되며 오유의 수도 감소되게 된다. 또한 디지크스트라는 다음과 같이 강조하였다. 《프로그람의 믿음성준위를 올리기 위한 유일한 효과적인 방도는 그것의 정확성에 대한 확실한 증명을

주는것이다.》[Dijkstra, 1972]. 그러나 제품이 정확하다고 증명되였다고 할지라도 그것은 철저히 시험되여야 한다. 시험을 정확성증명과 결합하여 진행해야 할 필요성을 설명하기 위하여 다음의 사실을 고찰해 보자.

1969년에 나우르는 제품을 구성하고 그 정확성을 증명하기 위한 기법에 대하여 보고하였다[Naur, 1969]. 이 기법은 나우르가 행편집문제(*line-editing problem*)이라고 명명한 문제에 의하여 례증되였다. 즉 오늘날에 와서 이 문제는 본문처리문제로 고찰되고 있다. 그에 대하여 다음과 같이 말할수 있다. 즉

본문은 공백문자나 행바꾸기, 문자에 의하여 분리된 단어로 이루어 진 본문이 주어 지게 되면 그것은 규칙에 따라서 행별형식으로 변환된다.

1. 행은 다만 공백문자나 행바꾸기문자가 있는 위치에서 끝난다.
2. 매행은 될수록 길게, 멀게 채워 진다.
3. 행에는 최대위치(maxpos) 이상의 문자들을 포함하지 않는다.

나우르는 이 기법을 리용하여 처리절차를 구성하고 비형식적으로 그것의 정확성을 증명하였다. 절차는 ALGOL60언어상에서 대략 25개 행으로 구성되였다. 이에 대한 론문은 그다음 레웬워스에 의해서 *Computing Reviews*잡지에서 검토되였다[Leavenworth, 1970]. 심사자는 나우르의 절차의 출력에서 첫 단어가 maxpos문자만큼 길지 않으면 첫행의 첫 단어는 공백으로 채워 진다고 지적하였다. 비록 이것이 사소한 오유인것 같지만 그것은 시험된 절차 즉 정확하다고 증명된 자료가 아니라 시험실례로서 실행된 절차로서 발견된 오유이다. 런던은 나우르의 절차에서 3개의 추가적인 오유를 발견하였다[London, 1971]. 하나는 이 절차가 maxpos문자보다 더 긴 단어를 만날 때까지 종결되지 않는다는것이다. 또한 이 오유도 이 절차가 수행되였다면 발견되였을것이다. 런던은 그다음 정확한 판본의 절차를 작성하고 그것이 정확하다는것을 형식적으로 증명하였다. 반면에 나우르는 비형식적인 증명기법만 리용하였던것이다.

다음의 일화는 구데나우와 게르하르트가 런던이 형식적인 《증명》을 하였음에도 불구하고 그가 발견하지 못했던 3개의 오유를 발견하였다는것이다[Goodenough and Gerhart, 1975]. 이러한 오유들에는 마지막단어의 뒤에 공백이나 행바꾸기문자가 놓이지 않으면 출력으로 될수 없다고 하는것도 있다. 다시 합리적인 시험실례를 선택하여 이 오유를 그리 어렵지 않게 발견하였다. 사실상 레웬워스와 런던 그리고 구데나우와 게르하르트가 발견한 7개의 오유들가운데서 4개는 나우르의 원래의 론문에서 보여 준 례증에서와 같은 시험실례를 실행시켜 간단히 발견하였다. 이로부터 찾게 되는 교훈은 명백하다. 제품은 정확하다고 증명되였다고 해도 여전히 철저히 시험하여야 한다는것이다.

6.5.1의 실례는 지어 작은 코드토막의 정확성증명이 오랜 과정으로 될수 있다는것을 보여 준다. 더우기 이 절의 실례연구는 25개행의 절차에서조차 정확성증명이 어렵고 오유를 범하기 쉽다는것을 보여 주었다. 그러므로 다음의 문제점이 제기된다. 즉 정확성증명이 흥미 있는 연구착상으로 되는가 혹은 시기적절한 위력한 쏘프트웨어공학기법인가

하는것이다. 이에 대한 대답은 다음절에서 보기로 한다.

6．5．3．정확성증명과 쏘프트웨어공학

많은 쏘프트웨어공학실천가들은 무엇때문에 정확성증명이 표준쏘프트공학기법으로 간주되지 말아야 하는가 하는 리유를 제기하고 있다. 첫째로 그것은 쏘프트웨어공학자들이 수학적으로 충분히 숙련되지 못했다는것이다. 둘째로 증명은 너무 비용이 들어서 실현할수 없다는것이다. 셋째로 증명은 아주 힘들다는것이다. 이러한 리유들은 지나치게 간소화되여 있다고 본다.

비록 6.5.1에서 보여 준 증명이 고등중학교 대수보다도 더 힘들게 리해될수는 없다고 하더라도 중요한 증명들은 입력명세서와 출력명세서, 순환불변량들이 1계술어계산 또는 2계술어계산 또는 그것과 등가인 계산으로 표현되여야 한다는것을 요구하고 있다. 이것은 다만 수학자들에게 있어서 증명과정을 더 간단하게 만들뿐아니라 정확성증명을 콤퓨터에서 진행할수 있게 한다. 문제를 좀더 복잡하게 한다면 술어계산이 지금에 와서 약간 뒤떨어 진것이라는것이다. 동시발생제품의 정확성을 증명하기 위해서는 시상론리나 기타 다른 양상론리를 리용한 기법이 필요된다[Manna and Pnueli, 1992]. 정확성증명이 수리론리에 대한 숙련을 요구한다는것은 의심할바 없다. 다행히도 오늘날 대부분의 콤퓨터과학전공부문들에서는 정확성증명기술을 전문으로 배우는것을 필수적인 과정으로 또는 배경으로 하고 있다는것이다. 그러므로 대학들에서는 지금 콤퓨터과학졸업생들에게 정확성증명을 위한 충분한 수학적지식을 주도록 하고 있다. 지난 시기에는 실천에 종사하는 쏘프트웨어공학자들은 수학적인 숙련을 반드시 필요로 하지 않는다는 주장이 사실이였지만 오늘날 해마다 수천명의 콤퓨터과학전공학생들이 산업에 망라되고 있는 실정에서는 그러한 주장을 더이상 할수 없게 되였다.

쏘프트웨어개발에서 증명을 리용하기에는 너무 비용이 든다고 하는 주장도 잘못되였다. 이와 대비적으로 정확성증명이 가지는 경제적인 생활력은 매 프로젝트의 기초는 비용 대 리득분석을 리용하는것이라는것으로써 알수 있다(5.2). 실례로 NASA우주정류소에 대한 쏘프트웨어를 생각해 보자. 인간이 위험에 처해 있고 만일 그 무엇인가 고장났다면 우주비행선구원단은 제때에 도착 못할수 있다. 생명에 결정적으로 관계되는 우주정류소 쏘프트웨어의 정확성을 증명하는데는 많은 비용이 든다. 그러나 정확성증명이 실행되지 않는 경우에 무시될수 있는 쏘프트웨어오유의 잠재적인 비용은 매우 크다.

세번째 주장은 정확성증명이 매우 어렵다는것이다. 이러한 주장이 있음에도 불구하고 조작체계의 핵심부와 콤파일러, 통신체계들을 비롯한 많은 중요한 제품들에 대하여 성과적으로 정확성이 증명되였다[Landwehr, 1983; Berry and Wing, 1985]. 더우기 정리증명프로그람과 같은 많은 도구들이 정확성증명에 리용되고 있다. 정리증명프로그람들은 제품과 그것의 입출력명세서와 순환불변량들을 입력으로 취한다. 그다음 정리증명프로그람들은 입력명세서를 만족시키는 자료를 입력하였을 때 출력명세서를 만족시키는 출력자료가 생성된다는것을 수학적으로 증명하게 된다.

이와 동시에 정확성증명과 관련한 일부 난점들도 있다. 실례로 정리증명프로그람이

정확하다는것을 어떻게 믿을수 있는가? 만일 정리증명프로그람이 《이 제품은 정확하다.》라고 인쇄하면 그것을 믿을수 있는가 하는것이다. 극단한 경우에 그림 6-7에 보여 준 이른바 정리증명프로그람을 생각해 보자.

아무리 코드가 이 정리증명프로그람에 복종한다고 해도 《이 제품은 정확하다.》고 인쇄할것이다. 달리 말하면 정리증명프로그람의 출력에 대하여 어떻게 믿을수 있는가? 한가지 제안은 바로 정리증명프로그람이 그자체에 복종하며 그것의 정확성여부를 보여 준다는것이다. 원리적인 의미를 떠나서 이것이 동작하지 않는다는것을 보여 주기 위한 간단한 방도는 그림 6-7에서 보여 준 정리증명프로그람이 증명에서 그자체에 복종한다고 하면 어떤 일이 일어 나겠는가 하는것을 고찰하는것이다. 언제나 그러하듯이 정리증명프로그람은 《이 제품은 정확하다.》고 인쇄하며 이로부터 《증명》은 그자체의 정확성을 증명하고 있다. 보다 더 난점으로 되는것은 입력명세서와 출력명세서 그리고 특히 양상론리와 같은 기타 다른 론리에서 순환불변량이나 그것의 등가물들을 찾아 내는것이다. 이제 제품이 정확하다고 하자. 만일 매 순환에 대하여 적당한 불변량을 찾을수 없으면 그 제품에 대하여 정확성을 증명할 방도는 없다. 도구들이 이러한 과제에 도움을 준다. 그러나 첨단도구를 가지고서도 쏘프트웨어공학자들은 정확성을 증명하지 못할수도 있다. 이러한 문제에 대하여 한가지 해결책은 6.5.2에서 주장한바와 같이 제품개발과 증명을 병렬적으로 진행하는것이다. 순환을 설계할 때 그 순환에 따르는 불변량은 동시에 규정된다. 이러한 방법으로 코드모듈이 정확하다는것을 증명하는것이 좀 더 쉽다.

```
void theoremProver()
{
    print "This product is correct";
}
```

그림 6-7. 《정리증명프로그람》

순환불변량을 찾아 낼수 없다고 하는것보다 더 나쁜 경우로서 만일 명세서 그자체가 부정확하다면 어떻게 되겠는가? 이것에 대한 실례가 방법 trickSort이다(그림 6-2). 그림 6-1의 부정확한 명세서가 주어 진 경우에 어떤 좋은 정리증명프로그람은 의심할바없이 그림 6-2에서 보여 준 조작이 정확하다고 선언할것이다.

만나와 왈딩거는 《우리는 절대로 명세서가 정확하다고 믿을수 없다.》, 《우리는 절대로 검증체계가 정확하다고 믿을수 없다.》라고 서술하였다[Manna and Waldinger, 1978]. 이와 같은 두명의 우수한 전문가들의 서술은 이전에 이야기된 여러가지 문제들을 모두 포함하고 있다.

이 모든것이 쏘프트웨어공학에서 정확성증명이 차지할 자리가 없다는것을 의미하는가? 그러나 그 답변은 이와 아주 대조적이다. 제품의 정확성을 증명하는것은 중요하며 때로는 사활적인 쏘프트웨어도구로 된다. 증명은 인간이 위기에 처하게 되는 곳에서 또는 반대로 비용 대 리득분석이 필요한 곳에서 적합하다. 만일 쏘프트웨어정확성을 증명

하는데 드는 비용이 제품이 실패한 경우에 드는 비용보다 더 적으면 그때는 제품에 대하여 증명해야 한다. 그러나 본문처리기의 경우의 연구에서 보여 준바와 같이 증명 그 하나만으로는 충분하지 않다. 대신에 정확성증명은 제품이 정확한가를 시험하는데 리용되는 그러한 기법모임의 중요한 구성부분으로 볼수 있다. 쏘프트웨어공학의 목적이 품질이 좋은 쏘프트웨어를 생산하는것이기때문에 정확성증명은 실지로 중요한 쏘프트웨어공학기법으로 된다. 지어 전체적으로 형식적인 증명이 정당하지 못할 때에도 쏘프트웨어의 품질은 비형식적인 증명을 리용하여 현저하게 개선될수 있다. 실례로 6.5.1에서와 류사한 증명은 순환이 정확한 회수만큼 진행된다는것을 검열하는데 도움을 줄수 있을것이다. 쏘프트웨어의 품질을 개선하기 위한 두번째 방도는 그림 6−6에서와 같이 코드에 단언을 삽입하는것이다. 만일 실행할 때 어떤 단언이 성립하지 않으면 제품은 개발이 중지될것이며 쏘프트웨어팀은 실행을 종결시킨 그 단언이 부정확한가 혹은 실지로 그 단언을 동작시킴으로써 발견된 코드에서의 오유가 실지로 존재하는가를 조사할수 있다. Eiffel[Meyer, 1992b]과 같은 언어들은 **assert**명령에 의하여 단언을 직접 지원한다. 이제 비형식적증명이 변수 xxx의 값이 코드의 특정한 곳에서 정수이여야 한다는것을 요구한다고 하자. 지어 설계팀이 xxx에 대하여 부수로 될수가 없다는것을 확인한다고 해도 더욱 믿을수 있도록 코드의 해당한 곳에 명령문

assert(xxx>0)

이 서술되여 있어야 한다는것을 규정할수도 있다. 만일 xxx가 0보다 작거나 같으면 실행은 끝나고 그다음 쏘프트웨어팀이 이런 상황을 조사할수 있다. 유감스럽게도 C++에서 ***Assert***는 C에서 ***assert***와 류사하게 하나의 오유수정명령문이다. 즉 그것은 언어 그자체의 일부분은 아니다. Ada95에서는 단언을 **pragma**로 서술하고 있다.

알고 싶은 문제

Java와 Ada(그러나 C, C++는 아니다.)와 같은 언어들에서 한가지 특징은 속박검사이다. 속박검사의 한가지 실례는 매 배렬첨수들이 그것이 선언된 범위안에 있다는것을 담보하는것을 실행기간에 시험하는것이다. 또 다른 실례는 부분적인 범위검사 즉 어떤 값이 어떤 변수에 할당될 때 그 값이 특정한 선언령역안에 있다는것을 실행할 때 검사하는것이다. 호아(Hoare)는 제품이 개발되는 기간에는 속박검사를 리용하고 일단 제품이 정확히 동작하면 그만두는것은 물이 없는 땅에서 구명옷을 입고 헤염치기를 배우고 실지 바다에 나가서는 구명옷을 벗어 버리는것과 류사할수 있다는것을 제안하였다. 호아는 자기가 1961년에 개발한 ALGOL60에 대한 콤파일러를 서술하였다[Hoare, 1981]. 사용자들에게 후에 이 콤파일러의 최종판본이 설치된후에 속박검사를 그만둘 기회가 제공되였을 때 그들은 한결같이 그것을 거절하였다. 왜냐하면 그들이 콤파일러의 초기판본의 실행중 검사하는 기간 값들이 범위를 벗어 나는 경우를 수많이 체험하였기때문이다.

속박검사는 보다 일반적인 개념인 단언검사의 특수한 경우로서 볼수 있다. 호아의 구명옷류추는 일단 최종판본이 설치되면 단언검사를 그만두는데 마찬가지로 리용될수 있다.

일단 사용자들이 제품이 정확하게 동작하고 있다고 확신하면 그들은 단언검사를 그만두도록 설정해야 한다. 이렇게 하면 실행은 빨리 진행되지만 단언검사가 중지된 경우에 단언에 의해서 발견될수 있는 오유들을 찾아 낼수 없게 된다. 그러므로 제품이 의뢰자의 콤퓨터에 설치된 다음이라도 실행시간에 대한 효과성과 단언검사를 계속하는 문제사이에는 절충을 해야 한다(이 문제에 대하여 흥미가 있다면 우의 《알고 싶은 문제》를 보시오).

실행에 기초한 시험에서 근본적인 문제는 쏘프트웨어개발팀성원들이 그 수행을 책임져야 한다는것이다.

6. 6. 실행에 기초한 시험은 누가 진행해야 하는가

이제 프로그람작성자가 자기가 작성한 모듈을 시험해 줄것을 부탁한다고 가정하자. 메이어는 시험을 오유를 발견할 목적으로 제품을 실행시키는 공정으로 서술하였다[Myers, 1979]. 그러므로 시험은 하나의 파괴과정이다. 다른 한편 시험을 진행하는 프로그람작성자는 보통 자기의 창조물을 파괴하려고 하지 않는다. 만일 코드에 대한 프로그람작성자의 기본태도가 일반적인 보호적인 태도라고 하면 프로그람작성자가 오유를 밝혀 줄 시험실례를 리용하는 기회는 기본동기가 실지 제품을 파괴하는것이였을 때보다 현저히 더 작다. 시험이 성공적이라는것은 오유를 찾는것이다. 이것 역시 난점을 가지고 있다. 이것은 이 모듈이 시험을 거치면 시험이 실패하였다는것을 의미한다. 반대로 모듈이 명세서에 따라 동작하지 않으면 시험은 성공하게 된다. 작성된 모듈에 대하여 시험해 줄것을 부탁받은 프로그람작성자는 실패(부정확한 동작)가 뒤따라 일어 나는 그런 방법으로 모듈을 실행시킬것을 요구한다. 이것은 프로그람작성자의 창조적인 본능과 대치된다.

결론은 불가피하게 프로그람작성자가 자기가 작성한 모듈을 시험하지 않도록 하는것이다. 프로그람작성자가 모듈을 작성한 다음에 모듈에 대한 시험은 창조자가 파괴적인 행동을 하도록 하고 그 창조물을 파괴하도록 할것을 요구한다. 실행에 기초한 시험이 그밖의 누군가에 의해서 진행되여야 할 두번째 리유는 바로 프로그람작성자가 설계나 명세서의 일부 측면에 대하여 잘못 리해하고 있을수도 있다는것이다. 만일 시험이 그밖의 누군가에 의하여 진행되면 그러한 오유는 발견될수도 있다. 그럼에도 불구하고 오유검사수정(실패의 원인을 찾고 그 오유를 정정한다.)은 그 코드에 제일 친숙한 사람인 원래의 프로그람작성자에 의하여 제일 잘 수행될수 있다.

프로그람작성자가 자기의 코드를 시험하지 말아야 한다는데 대해서는 더이상 언급하지 않는다. 이제 프로그람작성과정을 고찰해 보자. 프로그람작성자는 처음에 모듈에 대한 상세설계를 읽는다. 즉 이것은 흐름도표 또는 보다 좋기는 의사코드의 형식으로 될수도 있다. 그러나 어떤 기법을 리용하든지간에 모듈을 콤퓨터에 넣어 주기전에 철저히 탁상검열을 진행하여야 한다. 즉 프로그람작성자는 매 시험실례가 정확히 실행된다는것을 검사하기 위하여 상세설계를 추적하면서 여러가지 시험실례로써 흐름도표나 가상코드들을 검사해야 한다. 상세설계가 정확하다고 할 때만 본문편집기가 호출되고 모듈이 코드작성될것이다.

일단 모듈이 기계가독형식으로 작성되면 그것은 일련의 시험을 거치게 된다. 우선 프로그람작성자는 모듈콤파일을 시도한다. 이것이 성과적으로 진행되면 다음단계는 그것들을 련결하고 적재하는것이다. 그다음에는 모듈의 실행을 시도한다. 만일 모듈이 실행되면 모듈이 정확히 동작하는가를 결정하기 위하여 상세설계를 탁상검열할 때 리용한 시험실례와 같은 시험실례가 리용된다. 그다음 만일 모듈이 정확한 시험실례가 리용될 때 정확히 실행되면 프로그람작성자는 모듈의 로바스트성을 시험하기 위하여 부정확한 자료에 의한 시험을 시도한다. 모듈이 정확히 동작하면 체계적인 시험이 시작된다. 이것이 바로 프로그람작성자가 수행하지 말아야 하는 체계적인 시험(*systematic testing*)이다.

만일 프로그람작성자가 이러한 체계적인 시험을 진행하지 않는다면 과연 누가 그것을 진행하는가? 6.1.2에서 언급한바와 같이 SQA그룹에 의해서 독립적인 시험이 진행되여야 한다. 여기서 중요한것은 《독립적》이라는 단어이다. SQA그룹이 실지로 개발팀과 독립일 때에만 SQA그룹성원들은 그들의 작업을 저애할수 있는 제품최종기한과 같은 압력을 쏘프트웨어제품개발관리자들로부터 받음이 없이 제품이 실지로 명세서를 만족시킨다는것을 담보하기 위한 자기들의 사명을 수행할수 있다. SQA성원들은 자기들의 관리자에게 보고해야 하며 결국 자기들의 독립성을 지켜야 한다.

그러면 체계적인 시험은 어떻게 진행하는가? 시험실례에서 하나의 본질적인 부분은 시험이 실행되기전에 기대되는 출력에 대한 서술이다. 시험자가 말단에 앉아서 모듈을 실행시키고 시험실례를 아무렇게나 넣어 주고 그다음에 화면을 보면서 《내 추측에는 옳은것 같다.》라고 하는것은 완전히 시간랑비로 된다. 또한 시험자가 시험실례를 매우 세심하게 계획하고 매개의 시험실례를 차례로 실행시키고 출력을 지켜 보고 나서 《옳소, 확실히 정확해.》라고 하는것도 다같이 헛된 일이다. 만일 프로그람작성자가 자기들이 작성한 코드를 시험하도록 하면 프로그람작성자는 자기들이 기대하던것을 보게 되는 위험이 항상 존재한다. 그러한 위험은 시험이 그밖의 다른 사람에 의해서 실행될 때에도 발생할수 있다. 그에 대한 해결책은 관리자측이 시험을 진행하기전에 시험실례와 그에 대한 기대되는 결과자료를 다 기록해 두도록 하는것이다. 시험이 다 진행된 다음에 실제의 결과자료를 기록해 두고 기대되는 결과자료와 비교한다.

시험실례는 절대로 버리지 말아야 하기때문에 작은 기업체와 제품에 대해서도 기계가독형식으로 기록해 두는것이 중요하다. 그 리유는 바로 그것이 유지정비에서 필요하기때문이다. 제품이 유지정비되는 동안 회귀시험이 진행되여야 한다. 이미 제품이 정확히 실행되였다고 기록된 시험실례들은 축적되여 제품에 새로운 기능을 추가하기 위하여 만든 변경이 제품의 현존기능을 파괴하지 않는다는것을 담보하기 위하여 재실행되여야 한다. 이에 대해서는 16장에서 더 구체적으로 고찰하도록 한다.

6. 7. 언제 시험을 끝내는가

제품이 몇년동안 성과적으로 유지정비된후에 그 유용성이 마침내 없어 질수 있으며 전자관이 반도체소자로 교체된것처럼 전혀 다른 제품으로 교체될수 있다. 달리 말하면 제품이 아직은 쓸모 있을수도 있지만 그것을 새로운 하드웨어에 이식하거나 그것을 새로운

조작체계에서 동작시키는데 드는 비용은 새로 제품을 개발하는 비용보다 더 클수 있다. 그러므로 결국 쏘프트웨어제품은 페기되여 더이상 봉사되지 않게 된다. 쏘프트웨어가 어쩔수 없이 버려 지게 될 그 시점에서만 시험이 중지되게 된다.

이제는 필요한 배경자료를 모두 고찰하였으므로 객체들이 보다 자세히 고찰될수 있다. 이것은 다음장의 주제이다.

요 약

이 장의 중요한 화제는 시험이 쏘프트웨어개발공정의 모든 활동들과 병렬로 진행되여야 한다는것이다. 이 장은 품질문제에 대한 서술로 시작된다(6.1). 다음에는 관통심사회의와 검토에 대한 주의 깊은 론의와 함께 비실행에 기초한 시험이 서술된다(6.2). 그 다음에는 실행에 기초한 시험이 서술되고(6.3과 6.4) 유용성, 믿음성, 로바스트성, 성능 그리고 정확성을 비롯한 시험해야 할 제품의 동작특성이 론의된다(6.4.1부터 6.4.5). 6.5에서 정확성증명에 대하여 소개하고 이와 같은 증명의 실례를 6.5.1에 주었다. 그다음 쏘프트웨어공학에서 정확성증명의 역할이 분석된다(6.5.2와 6.5.3). 또 하나의 중요한 론점은 체계적인 실행에 기초한 시험이 프로그람작성자가 아니라 독립적인 SQA그룹에 의해서 진행되여야 한다는것이다(6.6). 마지막으로 시험이 언제 중지되는가 하는것을 론의하였다(6.7)

보 충

시험과정에 대한 쏘프트웨어개발자들의 태도는 시험을 제품이 몇년동안 정확히 동작하는가를 고찰하기 위한 수단으로 보는 태도로부터 시험이 요구사항확정과 명세작성, 설계 그리고 실현오유를 방지한다고 보는 현대적인 태도로 변화되였다. 이러한 전진에 대해서는 문헌 [Gelperin and Hetzel, 1988]에서 서술하고 있다. 쏘프트웨어시험의 본성과 그것이 왜 그렇게 어려운것인가 하는 리유에 대해서는 문헌 [Whittaker, 2000]에서 서술하고 있다.

문헌 [Tevonen, 1996]을 비롯하여 *IEEE Software* 1996년 1월호에는 쏘프트웨어의 품질과 관련한 기사들이 많이 게재되여 있다. *IEEE Computer* 1996년 11월호에도 쏘프트웨어의 품질에 대한 기사들이 들어 있다. *Communications of the ACM* 1997년 6월호에도 역시 쏘프트웨어의 품질에 관한 일련의 기사들이 들어 있다. 특별히 흥미를 끄는것은 쏘프트웨어개발공정을 개선하는것이 쏘프트웨어품질에 주는 영향을 서술한 문헌 [Herbsleb et al., 1997]과 McDonnel-Douglas에서 쏘프트웨어의 품질을 개선하기 위하여 전체적인 품질관리를 어떻게 리용하였는가에 대하여 서술한 문헌 [Arthur, 1997]이다. 품질을 개선하기 위한 기타 다른 방법들은 문헌 [Onoma and Yamaura, 1995]에 서술되여 있다. 쏘프트웨어제품의 전반적품질을 평가하는 방법은 문헌 [Boloix and Robillard, 1995]에서 서술하고 있다. 쏘프트웨어품질과 관련한 일화들은 문헌 [Voas, 1999]에 서술되여 있다. 믿음성은 *IEEE Software* 1995년 5월호 기사들에 서술되여 있다.

문헌 [Baber, 1987]에서는 프로그람의 정확성증명에 대하여 잘 소개하고 있다. 정확성증명의 표준기술의 하나는 호아의 론리라고 불리우는것을 리용하는것인데 [Hoare, 1969]에 서술되여 있다. 제품이 명세서를 만족시키고 있다는것을 담보해 주는 또 다른 하나의 방법은 매 단계에서 정확성을 유지하는가를 검사하면서 제품을 계단식으로 개발하는것이다. 이에 대해서는 문헌 [Dijkstra, 1969a, and, Wirth, 1971]에 서술되여 있다. 쏘프트웨어공학집단에서의 정확성증명과 관련한 중요한 론문이 문헌 [DeMillo, Lipton, and Perlis, 1979]에 서술되여 있다. IEEE《Standard for Software Reviews》[IEEE 1028, 1997]은 비실행에 기초한 시험에 대한 정보를 제공하는 중요한 문헌으로 되고 있다. 검토를 진행하는 방법이 그것의 효과성과 함께 문헌 [Ackerman, Buchwald, and Lewski, 1989]에 서술되여 있다. 대규모쏘프트웨어제품(25만코드행)에 대한 검토에 대해서는 문헌 [Russell, 1991]에 서술되여 있다. 검토와 관련된 경험에 대한 정보자료들은 문헌 [Doolan, 1992, and Weller, 1993]에 들어 있다. 팀의 규모를 줄이고 검토를 위해서 기다리게 되는 시간을 최소화하는것과 같은 비용에 관한 문제는 문헌 [Volta, 1993]에 서술되였다. 대규모쏘프트웨어의 검토를 진행하는데 드는 비용과 리득을 타산하는 실험은 문헌 [Porter, Siy, Toman, and Votta, 1997]에 서술되였다. 검토와 관련한 여러가지 유용한 론문들은 문헌 [Wheeler, Brykczynski, and Meeson, 1996]에서 찾아 볼수 있다. 검토령역에서의 앞으로 가능한 연구개발에 대해서는 문헌 [Johnson, 1998]에서 제기하였다.

실행에 기초한 시험에 대한 고전적인 저서는 문헌 [Myers, 1979]이다. 이 저서는 시험분야에 큰 영향을 주는 문헌으로 되고 있다. 문헌 [DeMillo, Lipton, and Sayward, 1978]은 시험자료의 선택과 관련한 정보들을 서술한 훌륭한 문헌으로 되고 있다. 류사한 문헌 [Beizer, 1990]은 시험에 대한 개론, 좋은 편람으로 되고 있다. 이와 류사한 저서는 문헌 [Hetzel, 1988]이다.

특히 객체지향파라다임에 대하여 보면 *Communications of the ACM* 1994년 9월호에는 객체지향쏘프트웨어의 시험과 관련한 수많은 기사들이 게재되여 있다. 문헌 [Kung, Hsia and Gao, 1998]은 객체지향시험과 관련한 책이며 문헌 [Sykes and McGregor, 2000]도 마찬가지이다.

IEEE Software 1989년 5월호에는 시험문제와 관련한 폭 넓은 기사들이 게재되여 있다. 실행 및 비실행에 기초한 시험이 모두 포함되여 있다. 쏘프트웨어시험 및 분석과 관련한 국제토론회회보들에는 시험문제와 관련하여 우와 류사하게 넓은 범위에서 서술하고 있다. *Communications of the ACM* 1997년 4월호에서는 오유수정검사에 대하여 서술하였다.

문 제

6.1. 용어 정확성증명과 검증, 타당성이 이 책에서는 어떻게 리용되고 있는가?

6.2. 어느 한 쏘프트웨어개발기업체에는 쏘프트웨어의 검증과 타당성이 물론 개발을 진행하는 17명의 관리자를 비롯한 83명의 쏘프트웨어전문가들에게 있다. 최근의 통계는 그들이 28%의 시간을 검증과 타당성증명에 돌리고 있다는것을 보여 주고 있

다. 어떤 관리자의 회사에 대한 년 평균비용은 14만딸라이고 반면에 비관리성원인 전문가에 대해서는 년 평균 1만 4천딸라를 지출한다. 즉 우의 두 수자는 다 총적값을 포함하고 있다. 비용 대 리득분석을 리용하여 기업체안에 SQA그룹을 따로 두겠는가를 결정하시오.

6.3. 두명의 관리성원을 포함하여 다만 7명의 전문가들로 구성된 회사에 대하여 련습문제 6.2에서 제기한 비용 대 리득분석을 반복하시오. 기타 다른 수자들에 대해서는 달라진것이 없다고 가정하시오.

6.4. 한개의 모듈을 11일동안 시험하여 두개의 오유를 발견하였다. 이것은 기타 다른 오유들의 존재에 대하여 무엇을 말해 주는가?

6.5. 관통심사회의와 검토사이의 류사한 점은 무엇이며 차이점은 무엇인가?

6.6. 당신은 Ye Olde Fashioned Software에 있는 SQA그룹의 한성원이다. 당신은 관리자에게 검토를 도입해야 한다고 제기하였다. 그는 같은 코드부분에 대하여 한 사람이 시험실례를 실행할수 있는 경우에 오유를 찾기 위하여 4명의 사람이 시간을 랑비할 리유는 없다고 대답했다. 당신은 무엇이라고 대답하겠는가?

6.7. 당신들은 모두가 텍사스와 테네쎄에 모두 154대의 지점을 가진 세금준비독점체인 텍사스세무소의 SQA성원들이다. 독점체의 소유자들은 기업전반에서 리용할 새로운 형태의 세금준비쏘프트웨어패키지를 구입하려고 생각하고 있다. 그 쏘프트웨어를 허가하기전에 당신에게 그것을 철저히 시험할 임무가 맡겨 졌다. 이 쏘프트웨어의 어떤 특성들을 조사하겠는가?

6.8. 텍사스세무소의 154개의 지점들은 모두 통신망으로 련결되여 있다. 판매대표는 당신에게 자기가 팔려고 하는 통신쏘프트웨어를 4주동안 마음대로 시험해 보도록 하였다. 그 쏘프트웨어에 대하여 어떤 시험을 진행하여야 하며 왜 그런가를 설명하시오.

6.9. 당신은 새로운 함선대함선미싸일을 조종하기 위한 쏘프트웨어를 개발할 책임을 진 클라몬트공화국의 해군소장이다(문제 1.3). 쏘프트웨어가 당신에게 인수시험을 위하여 배포되였다. 쏘프트웨어에 대하여 어떤 특성을 시험해야 하는가?

6.10. 만일 순환불변량

$$s = y[0] + y[1] + \cdots + y[k-1]$$

이 식(6.4)대신에 리용된다고 하면 6.5.1의 정확성증명에 대하여 어떤 일이 벌어 지게 되는가?

6.11. 당신은 순환불변량에 대하여 얼마간의 경험을 가지고 있고 불변량(6.4)이 그림 6-6에 있는 순환에 대한 정확한 불변량이라는것을 알고 있다고 가정하자. 출력명세서(6.3)가 순환불변량에 대하여 본질적인 결과로 된다는것을 밝히시오.

6.12. 다음의 코드토막을 고찰해 보시오.

```
k = 0;
g = 1;
while (k < n)
{
  k = k + 1;
  g = g * k;
}
```

만일 n이 정수이면 이 코드토막은 g = n!을 정확히 계산해 낸다는것을 증명하시오.

6.13. 의뢰자에게 배포된 제품이 실지로 의뢰자가 요구한것이 아니라고 하는 경우에 이 문제에 대하여 정확성검사를 진행할수 있는가? 리유를 밝히시오.

6.14. 디지크스트라의 명령문(6.3)을 시험이 아니라 정확성증명에 리용하자면 어떻게 변화시켜야 하는가? 6.5.2의 실례연구를 명심하시오.

6.15. 교원이 지정한 언어를 리용하여 나우르의 본문처리문제(6.5.2)에 대한 해결방안을 설계하고 실현하시오. 시험자료에 대하여 그것을 실행하고 발견한 오유의 수와 매 오유의 원인(실례로 론리적인 오유, 순환결수오유 등)을 기록하시오. 발견된 오유들을 모두 정정하지 마시오. 이제 제품들을 다른 학생들과 교환하여 매 사람이 다른 제품에서 발견한 오유들이 얼마나 되고 그것들은 새로운 오유인가 아닌가를 살펴 보시오. 그리고 오유들의 원인을 기록하고 찾은 오유류형를 서로 비교하시오. 부류별로 결과들을 표로 작성하시오.

6.16. (과정안상 목표) 부록 1에 있는 브로드랜즈지역 아동병원쏘프트웨어제품에 대하여 유용성, 믿음성, 로바스트성, 성능, 정확성을 어떻게 시험하겠는가에 대하여 설명하시오.

6.17. (쏘프트웨어공학독본) 교원은 문헌 [Whittaker, 2000]의 복제본을 배포해 줄것이다. 브룩스(2.9)는 쏘프트웨어의 4가지 곤난이 본질적으로 어렵다고 생각하고 있다. 시험이 본질적으로 어렵다고 또는 우연적으로 어렵다고 생각하는가?

참 고 문 헌

[Ackerman, Buchwald, and Lewski, 1989] A. F. ACKERMAN, L. S. BUCHWALD, AND F. H. LEWSKI, "Software Inspections: An Effective Verification Process," *IEEE Software* **6** (May 1989), pp. 31–36.

[Arthur, 1997] L. J. ARTHUR, "Quantum Improvements in Software System Quality," *Communications of the ACM* **40** (June 1997), pp. 46–52.

[Baber, 1987] R. L. BABER, *The Spine of Software: Designing Provably Correct Software: Theory and Practice,* John Wiley and Sons, New York, 1987.

[Beizer, 1990] B. BEIZER, *Software Testing Techniques,* 2nd ed., Van Nostrand Reinhold, New York, 1990.

[Berry and Wing, 1985] D. M. BERRY AND J. M. WING, "Specifying and Prototyping: Some Thoughts on Why They Are Successful," in: *Formal Methods and Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development,* Volume 2, Springer-Verlag, Berlin, 1985, pp. 117–28.

[Boehm, 1984a] B. W. BOEHM, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software* **1** (January 1984), pp. 75–88.

[Boloix and Robillard, 1995] G. BOLOIX AND P. N. ROBILLARD, "A Software System Evaluation Framework," *IEEE Computer* **28** (December 1995), pp. 17–26.

[Bush, 1990] M. BUSH, "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory," *Proceedings of the 12th International Conference on Software Engineering,* Nice, France, March 1990, pp. 196–99.

[DeMillo, Lipton, and Perlis, 1979] R. A. DEMILLO, R. J. LIPTON, AND A. J. PERLIS, "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM* **22** (May 1979), pp. 271–80.

[DeMillo, Lipton, and Sayward, 1978] R. A. DEMILLO, R. J. LIPTON, AND F. G. SAYWARD, "Hints on Test Data Selection: Help for the Practicing Programmer," *IEEE Computer* **11** (April 1978), pp. 34–43.

[Dijkstra, 1968] E. W. DIJKSTRA, "A Constructive Approach to the Problem of Program Correctness," *BIT* **8** (No. 3, 1968), pp. 174–86.

[Dijkstra, 1972] E. W. DIJKSTRA, "The Humble Programmer," *Communications of the ACM* **15** (October 1972), pp. 859–66.

[Doolan, 1992] E. P. DOOLAN, "Experience with Fagan's Inspection Method," *Software—Practice and Experience* **22** (February 1992), pp. 173–82.

[Fagan, 1976] M. E. FAGAN, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15** (No. 3, 1976), pp. 182–211.

[Fagan, 1986] M. E. FAGAN, "Advances in Software Inspections," *IEEE Transactions on Software Engineering* **SE-12** (July 1986), pp. 744–51.

[Fowler, 1986] P. J. FOWLER, "In-Process Inspections of Workproducts at AT&T," *AT&T Technical Journal* **65** (March/April 1986), pp. 102–12.

[Garman, 1981] J. R. GARMAN, "The 'Bug' Heard 'Round the World," *ACM SIGSOFT Software Engineering Notes* **6** (October 1981), pp. 3–10.

[Gelperin and Hetzel, 1988] D. GELPERIN AND B. HETZEL, "The Growth of Software Testing," *Communications of the ACM* **31** (June 1988), pp. 687–95.

[Goodenough, 1979] J. B. GOODENOUGH, "A Survey of Program Testing Issues," in: *Research Directions in Software Technology,* P. Wegner (Editor), The MIT Press, Cambridge, MA, 1979, pp. 316–40.

[Goodenough and Gerhart, 1975] J. B. GOODENOUGH AND S. L. GERHART, "Toward a Theory of Test Data Selection," *Proceedings of the Third International Conference on Reliable Software,* Los Angeles, 1975, pp. 493–510. Also published in *IEEE Transactions on Software Engineering* **SE-1** (June 1975), pp. 156–73. Revised version: J. B.

Goodenough, and S. L. Gerhart, "Toward a Theory of Test Data Selection: Data Selection Criteria," in: *Current Trends in Programming Methodology,* Volume 2, R. T. Yeh (Editor), Prentice Hall, Englewood Cliffs, NJ, 1977, pp. 44–79.

[Herbsleb et al., 1997] J. HERBSLEB, D. ZUBROW, D. GOLDENSON, W. HAYES, AND M. PAULK, "Software Quality and the Capability Maturity Model," *Communications of the ACM* **40** (June 1997), pp. 30–40.

[Hetzel, 1988] W. HETZEL, *The Complete Guide to Software Testing,* 2nd ed., QED Information Systems, Wellesley, MA, 1988.

[Hoare, 1969] C. A. R. HOARE, "An Axiomatic Basis for Computer Programming," *Communications of the ACM* **12** (October 1969), pp. 576–83.

[Hoare, 1981] C. A. R. HOARE, "The Emperor's Old Clothes," *Communications of the ACM* **24** (February 1981), pp. 75–83.

[IEEE 610.12, 1990] "A Glossary of Software Engineering Terminology," IEEE 610.12-1990, Institute of Electrical and Electronic Engineers, New York, 1990.

[IEEE 1028, 1997] "Standard for Software Reviews," IEEE 1028, Institute of Electrical and Electronic Engineers, New York, 1997.

[Johnson, 1998] P. M. JOHNSON, "Reengineering Inspection," *Communications of the ACM* **42** (February 1998), pp. 49–52.

[Jones, 1978] T. C. JONES, "Measuring Programming Quality and Productivity," *IBM Systems Journal* **17** (No. 1, 1978), pp. 39–63.

[Kelly, Sherif, and Hops, 1992] J. C. KELLY, J. S. SHERIF, AND J. HOPS, "An Analysis of Defect Densities Found during Software Inspections," *Journal of Systems and Software* **17** (January 1992), pp. 111–17.

[Kung, Hsia, and Gao, 1998] D. C. KUNG, P. HSIA, AND J. GAO, *Testing Object-Oriented Software,* IEEE Computer Society Press, Los Alamitos, CA, 1998.

[Landwehr, 1983] C. E. LANDWEHR, "The Best Available Technologies for Computer Security," *IEEE Computer* **16** (July 1983), pp. 86–100.

[Leavenworth, 1970] B. LEAVENWORTH, Review #19420, *Computing Reviews* **11** (July 1970), pp. 396–97.

[London, 1971] R. L. LONDON, "Software Reliability through Proving Programs Correct," *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing,* March 1971.

[Manna and Pnueli, 1992] Z. MANNA AND A. PNUELI, *The Temporal Logic of Reactive and Concurrent Systems,* Springer-Verlag, New York, 1992.

[Manna and Waldinger, 1978] Z. MANNA AND R. WALDINGER, "The Logic of Computer Programming," *IEEE Transactions on Software Engineering* **SE-4** (1978), pp. 199–229.

[Meyer, 1992b] B. MEYER, *Eiffel: The Language,* Prentice Hall, New York, 1992.

[Myers, 1979] G. J. MYERS, *The Art of Software Testing,* John Wiley and Sons, New York, 1979.

[Naur, 1969] P. NAUR, "Programming by Action Clusters," *BIT* **9** (No. 3, 1969), pp. 250–58.

[New, 1992] R. NEW, personal communication, 1992.

[Onoma and Yamaura, 1995] A. K. ONOMA AND T. YAMAURA, "Practical Steps toward Quality Development," *IEEE Software* **12** (September 1995), pp. 68–77.

[Porter, Siy, Toman, and Votta, 1997] A. A. PORTER, H. P. SIY, C. A. TOMAN, AND L. G. VOTTA, "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies," *IEEE Transactions on Software Engineering* **23** (March 1997), pp. 129–45.

[Russell, 1991] G. W. RUSSELL, "Experience with Inspection in Ultralarge-Scale Developments," *IEEE Software* **8** (January 1991), pp. 25–31.

[Sykes and McGregor, 2000] D. A. SYKES AND J. D. McGREGOR, *Practical Guide to Testing Object-Oriented Software,* Addison-Wesley, Reading, MA, 2000.

[Tevonen, 1996] I. TEVONEN, "Support for Quality-Based Design and Inspection," *IEEE Software* **13** (January 1996), pp. 44–54.

[Voas, 1999] J. VOAS, "Software Quality's Eight Greatest Myths," *IEEE Software* **16** (September/October 1999), pp. 118–120.

[Volta, 1993] L. G. VOLTA, JR., "Does Every Inspection Need a Meeting?" *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM SIGSOFT Software Engineering Notes* **18** (December 1993), pp. 107–14.

[Weller, 1993] E. F. WELLER, "Lessons from Three Years of Inspection Data," *IEEE Software* **10** (September 1993), pp. 38–45.

[Wheeler, Brykczynski, and Meeson, 1996] D. A. WHEELER, B. BRYKCZYNSKI, AND R. N. MEESON, JR., *Software Inspection: An Industry Best Practice,* IEEE Computer Society, Los Alamitos, CA, 1996.

[Whittaker, 2000] J. A. WHITTAKER, "What Is Software Testing? And Why Is It So Hard?" *IEEE Software* **17** (January/February 2000), pp. 70–79.

[Wirth, 1971] N. WIRTH, "Program Development by Stepwise Refinement," *Communications of the ACM* **14** (April 1971), pp. 221–27.

제７장. 모듈로부터 객체에로

일부 콤퓨터잡지들은 객체지향파라다임이 1980년대 중엽의 돌발적이고 극적인 새로운 발견이며 당시에 인기 있던 구조화파라다임에 대처한 혁신적인 선택이였다고 암시하였다. 그러나 사실은 그렇지 않다. 모듈성에 대한 리론은 1970년대와 1980년대에 끊임 없는 발전을 이룩하였으며 객체는 모듈성리론의 테두리안에서의 단순한 론리적인 발전이였다(다음의 《알고 싶은 문제》를 보시오.).

> **알고 싶은 문제**
>
> 객체지향개념들은 모의언어 Simula 67 [Dahl and Nygaard, 1966]로 1966년에 소개되였다. 그러나 그때는 그 기능이 실용화되기에는 너무 일렀다. 그래서 1980년대 초까지 그것은 표현화되지 않았다. 1980년대 초에 그것은 본질상 모듈성리론의 범위안에서 재발명되였다.
>
> 이 장의 다른 실례들은 세계가 선진기술을 리용할수 있도록 준비될 때까지 그 기술이 활성화되지 않는 방식이 존재한다는것을 말해 주고 있다. 실례로 정보은페(7.6)는 1971년에 파나스에 의하여 쏘프트웨어의 범위내에서 처음으로 제기되였다[Parnas, 1971]. 그러나 그 기술은 약 10년후 즉 교갑화와 추상자료형들이 쏘프트웨어공학의 부분으로 되였을 때까지는 널리 일반화되지 못하였다. 인간은 반드시 새로운 착상이 처음 나왔을 때가 아니라 그것들을 리용할 준비가 되여 있을 때에만 받아 들이는것 같다.

이 장에서는 모듈성의 범위내에서 객체를 서술하였다. 객체지향파라다임이 구조화파라다임보다 왜 더 우월한가를 리해하지 못하고서는 객체를 정확히 리용하는것이 매우 어렵기때문에 이러한 방법을 취하였다. 그리고 이것을 위해서는 객체가 다만 모듈의 개념으로부터 시작되고 지식의 체계안에서 모듈의 다음단계로 된다는것을 정확히 리해하는것이 필요하다.

7．1．모듈이란 무엇인가

큰 제품이 하나의 유일한 코드블로크로 이루어 져 있을 때 유지정비는 상상할수 없을 정도로 어렵다. 이런것을 만든 사람에게 있어서도 코드를 오유수정하기 위하여 노력한다는것은 대단히 어려운 일이다. 또 다른 프로그람작성자가 그것을 리해한다는것은 사실상 불가능하다. 해결책은 그 제품을 모듈이라고 부르는 보다 작은 쪼각들로 분할하는것이다. 모듈이란 무엇인가? 제품을 모듈로 분할하는 방식 그자체가 중요한가 아니면 큰 제품을 보다 작은 코드쪼각으로 분할하는것이 중요한가?

일찌기 스티븐스(Stevens), 마이어스(Myers)와 콘스탄틴(Constantine)이 이와 같은 모

듈에 대하여 설명하려고 시도하였다. 그들은 모듈은《체계의 다른 부분들이 그것을 불러 낼수 있는 이름을 가지며 더 좋기는 자기의 고유하고 명백한 변수이름들의 모임을 가지는 한개 또는 그이상의 련관된 프로그람명령들의 모임》이라고 정의하였다[Stevens, Myers, and Constantine, 1974]. 다른 말로 모듈은 처리절차, 함수, 방법을 호출하는 방식으로 호출할수 있는 한개의 간단한 코드블로크들로 이루어 져 있다. 이러한 정의는 극히 일반적인것 같다. 그것은 내부적이든 개별적이든 콤파일되는 모든 종류의 처리절차와 함수들을 포함하고 있다. 그것은 비록 자기의 고유한 변수들을 가질수 없다고 해도 COBOL단락들과 부분들을 포함하고 있다. 왜냐하면 이 정의는 변수이름들의 명백한 모임을 가지고 있다는 특성이 선택적이라는것을 말해 주고 있기때문이다. 그것은 또한 다른 모듈들에 삽입되는 모듈들을 포함하고 있다. 그러나 이 정의는 일반적이기는 하지만 그리 완전한것은 아니다. 실례로 아쎔블리어마크로는 불러 내지 못하며 따라서 그것은 앞의 정의에 따르면 모듈이 아니다.

C언어와 C++언어에서는 류사하게 제품안에 #**include**로 정의된 선언들의 머리부파일들은 불러 내지 못한다. 즉 Ada패키지(추상자료형의 실현)라든가 Ada일반명(마크로)도 불러 내지 못한다. 한마디로 이 정의는 너무 제한적이다.

요르돈(Yourdon)과 콘스탄틴(Constantine)은 보다 일반적인 정의를 주었다. 즉《모듈은 말그대로 경계요소에 의하여 제한되고 집합식별자를 가지고 있으며 어휘상 린접되여 있는 프로그람명령문들의 렬이다.》[Yourdon and Constantine, 1979]. 경계요소의 실례로서 Pascal이나 Ada와 같은 블로크구조화된 언어들에서의 **begin**…**end**쌍과 C++ 혹은 Java언어들에서의 {...}쌍을 들수 있다. 이 정의는 앞선 정의에서 제기된 모든 경우들을 포함할뿐아니라 포괄범위가 매우 넓으므로 이 책의 전반에서 리용되고 있다. 특히 고전적인 파라다임에서의 처리절차와 함수들은 모듈이다. 객체지향파라다임에서 객체는 모듈이고 객체안에서의 방법도 역시 모듈이다.

모듈화의 중요성을 리해하기 위하여 어느 정도 공상적인 다음의 실례들을 고찰하자. 죤 헨스(John Hence)는 아주 무능한 콤퓨터설계가였다. 그는 아직도 **NAND**문회로와 **NOR**문회로가 완비성을 가진다는것 즉 모든 회로들을 **NAND**문회로나 **NOR**문회로만 가지고도 만들수 있다는것을 발견하지 못했다. 따라서 죤은 **AND**, **OR** 그리고 **NOT**문회로를 리용하여 산수론리장치와 밀기등록기, 16개의 등록기를 만들기로 결심하였다. 그 결과로서 만들어 진 콤퓨터를 그림 7-1에 보여 주었다. 여기서는 세개의 구성요소들이 단순한 방식으로 서로 련결되여 있다. 한 설계가는 세개의 규소소편으로 회로를 만들어야 한다고 결심하고 그림 7-2에 보여 준바와 같이 세개의 소편들로 설계를 진행하였다. 한개의 소편은 산수론리장치의 모든 문회로들을 가지고 있고 두번째 소편에는 밀기등록기가 들어 있으며 세번째 소편은 등록기를 위한것이다. 이 시점에서 죤은 술집에서 누군가가 자기에게 오직 한가지 종류의 문회로로 된 소편들을 만드는것이 가장 좋다고 말한것이 어렴풋이 생각나서 소편들을 다시 설계하였다. 그는 소편 1에는 모두 **AND**문회로들을 넣고 소편 2에는 모두 **OR**문회로들을 넣었으며 소편 3에는 모두 **NOT**문회로를 넣도록 하였다.

그 결과물을 그림 7-3에 도식적으로 보여 주었다.

그림 7-2와 7-3은 기능상 같다. 즉 정확히 말하여 그것들은 꼭 같은 일을 한다. 그러나 이 설계들은 뚜렷한 서로 다른 특성을 가지고 있다.

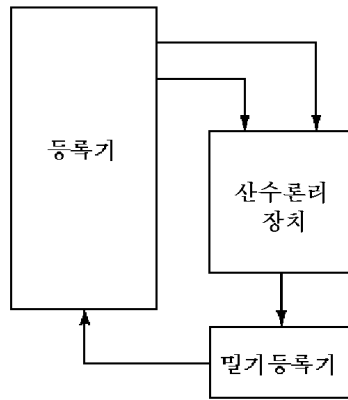



그림 7-1. 콤퓨터의 설계

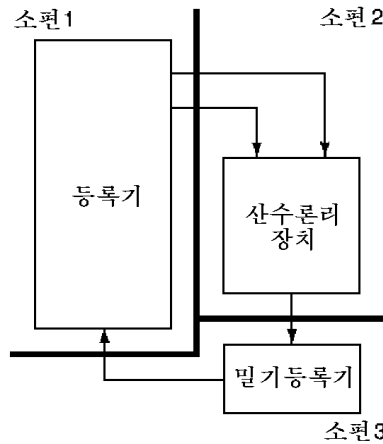



그림 7-2. 세개의 소편으로 조립한 그림 7-1의 콤퓨터

첫째로, 그림 7-3은 그림 7-2보다 대단히 리해하기 어렵다. 수자론리에 대한 지식이 있는 대부분의 사람들은 인차 그림 7-2에 있는 소편들이 산수론리장치, 밀기등록기, 등록기들의 모임을 형성한다는것을 알게 될것이다. 그러나 하드웨어전문가들조차도 그림 7-3에 있는 여러가지 **AND**, **OR**, **NOT** 문회로의 기능을 리해하기 힘들것이다.

둘째로, 그림 7-3에 있는 회로들에 대한 교정유지정비는 오유를 범하게 될것이다. 즉 결함이 어디에 있는가를 결정하는것이 어려울것이다. 다른 한편 그림 7-2에 있는 콤퓨터의 설계에 결함이 있다면 그 결함이 산수론리장치가 동작하는 방식, 밀기등록기가 동작하는 방식 혹은 등록기가 동작하는 방식에 있겠는가를 결정함으로써 그 결함의 위치를 알아 낼수 있다. 이와 류사하게 만일 그림 7-2의 콤퓨터가 파괴된다면 어느 소편을 교체할것인가를 결정하는것은 상대적으로 쉽다. 그리고 만일 그림 7-3에 있는 콤퓨터가 파괴된다면 아마 세개의 소편들을 모두 교체하는것이 좋을것이다.

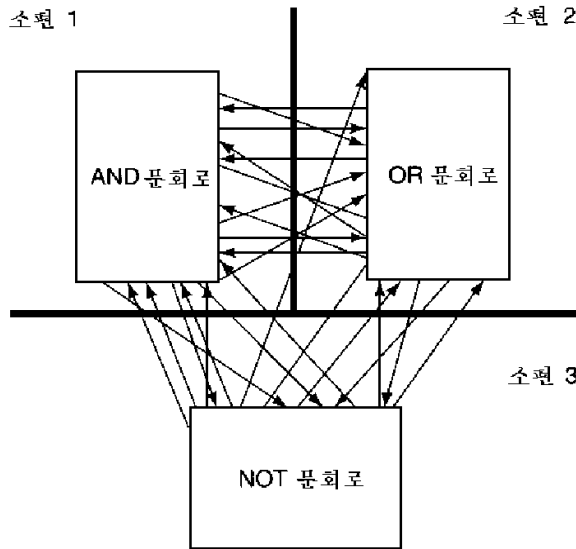



그림 7-3. 세개의 소편으로 조립한 그림 7-1의 콤퓨터

셋째로, 그림 7-3의 콤퓨터는 확장 또는 보강하기가 힘들다. 만일 새형의 산수론리장치가 필요된다든가 혹은 보다 빠른 등록기들이 요구된다면 처음부터 다시 설계하여야 한다. 그러나 그림 7-2의 콤퓨터의 설계는 적절한 소편을 쉽게 교체할수 있게 한다. 아마도 가장 나쁜 점은 그림 7-3의 소편들이 임의의 새로운 제품들에서 다시 리용될수 없다는것이다. **AND**, **OR**, **NOT**문회로들의 특정한 조합은 그것들이 설계된 제품이외의 다른 임의의 제품들에서 리용될수 있다고는 말할수 없다. 모든 가능성으로 보아 그림 7-2의 세가지 소편들은 산수론리장치, 밀기등록기 또는 등록기들을 요구하는 다른 제품들에서 다시 리용할수 있다.

여기서 문제점은 쏘프트웨어제품들이 그림 7-2에서처럼 개별적소편안에서는 최대한의 관계를 가지고 소편들사이에서는 최소한의 관계를 가지도록 설계되여야 한다는것이다. 모듈은 그것이 한가지 동작 혹은 련속적인 동작들을 수행하는 다른 모듈들과 련결되여 있다는 점에서 소편들에 비유할수 있다. 전체적인 제품의 기능은 고정되여 있다. 결정해야 할것은 제품을 어떻게 모듈로 분할하는가 하는것이다. 1장에서 지적한것처럼 합성/구조화된 설계[Stevens, Myers, and Constantine, 1974]는 유지정비비용과 전체 쏘프트웨어예산의 주요한 구성요소들을 줄이기 위한 방도로서 제품을 모듈들로 분할하기 위한 론리적인 근거를 준다. 유지정비의 노력은 점점 그것이 정정이든, 완성이든, 적응이든 개별적모듈안에서는 최대한도의 호상작용이, 모듈들사이에서는 최소한도의 호상작용이 있을 때 줄어 들게 된다. 달리 말하여 합성/구조화된 설계(C/SD)의 목적은 제품의 모듈분해가 그림 7-3이 아니라 그림 7-2를 조립한다는것을 담보하는것이다. 마이어스는 모듈안에서의 호상작용정도를 나타내는 모듈의 응집도(*cohesion*)와 두 모듈사이의 호상작용정도를 나타내는 모듈의 결합도(*coupling*)에 대한 착상을 하였다[Myers, 1978b]. 더 정확히 말하면 마이어스는 응집도가 아니라 강도(*strength*)라는 용어를 리용하였다. 그러나 응집도이라는

용어가 용어 강도보다 더 낫다. 왜냐하면 모듈이 고강도 또는 저강도를 가질수 있고 또 저강도라는 표현에 무엇인가 선천적인 모순 즉 무엇인가 강하지 못하면 약하다고 하는 모순이 있기때문이다. 일부 저자들은 결합이라는 표현대신에 속박이라는 표현을 리용하고 있다. 유감스럽게도 속박은 어떤 값들이 변수들에 속박된다는것과 같이 콤퓨터과학의 다른 분야에서도 리용되고 있다. 그러나 결합은 이런 함축된 의미를 가지고 있지 않으며 따라서 속박보다 더 낫다.

이 점에서 모듈의 작용, 모듈의 론리, 모듈의 문맥을 서로 구별하는것이 필요하다. 모듈의 작용(*action*)은 그것이 무엇을 하는가 하는 거동(*behavior*)이다. 실례로 모듈 m의 작용이 자기 인수들의 2차뿌리를 계산하는것을 들수 있다. 모듈의 론리(*logic*)는 모듈이 자기의 작용을 어떻게 수행하는가 하는것이다. 즉 모듈 m의 경우에 2차뿌리를 계산하는 특정한 방법은 뉴톤의 방법이다[Gerald and Wheatley, 1999]. 모듈의 문맥(*context*)은 그 모듈의 특정한 리용(*usage*)이다. 실례로 모듈 m은 배정확도옹근수의 2차뿌리를 계산하기 위하여 리용된다고 하자. C/SD에서의 중요한 점은 어떤 모듈에 할당된 이름은 그 모듈의 작용이며 그것의 론리나 그것의 문맥으로 되지 않는다는것이다. 그렇기때문에 C/SD에서 모듈 m은 compute square root(2차뿌리를 계산하시오.)라고 이름 지어야 하며 이 모듈의 론리와 문맥은 이름의 견지에서 보면 상관이 없다.

7. 2. 응집도

마이어스는 응집도를 7개의 범주 혹은 준위로 정의하였다 [Myers, 1978b]. 현대콤퓨터과학의 견지에서 볼 때 마이어스의 첫 두개 준위들을 완전히 같은것으로 간주된다. 보다 정확하게는 앞으로 보게 되는바와 같이 기능적인 응집도는 구조화파라다임에 대하여 최량적이며 반면에 정보적인 응집도는 객체지향파라다임에 대하여 최량적이다.

7.	정보적인 응집도 (좋다.) 기능적인 응집도
5.	통신적인 응집도
4.	절차적인 응집도
3.	림시적인 응집도
2.	론리적인 응집도
1.	일치적인 응집도 (나쁘다.)

그림 7-4. 응집도의 준위

결과적인 순서배렬을 그림 7-4에 보여 주었다. 이것은 임의의 정렬에 대한 선형적인 등급이 아니다. 그것은 단지 상대적인 순서배렬 즉 어느 형식의 응집도가 높고(좋고) 어

느 형식의 응집도가 낮은가(나쁜가) 하는것을 결정하기 위한 한가지 방법인것이다.

무엇이 높은 응집도를 가진 모듈을 이루는가 하는것을 리해하자면 다른 끝에서 시작하여 보다 낮은 응집도준위들을 고찰하는것이 필요하다.

7. 2. 1. 일치적인 응집도

모듈은 만일 그것이 전혀 관련이 없는 여러가지 작용들을 수행한다면 일치적인 응집도를 가진다. 일치적인 응집도를 가진 모듈의 실례는 다음행을 인쇄한다든가 두번째 인수를 구성하는 문자들의 렬을 반대로 놓는다든가 5번째 인수에 7을 더한다든가, 4번째 인수를 류동소수점수로 변환한다든가 등이다. 한가지 명백한 질문은 이러한 모듈들이 실천적으로 어떻게 제기될수 있겠는가 하는것이다. 가장 보편적인 리유는 《**모든 모듈들은 35~50개의 실행가능한 명령문들로 이루어 진다.**》와 같은 규칙을 엄격히 실시한 결과에 있다. 만일 쏘프트웨어기업체가 모듈이 너무 크거나 너무 작지 말아야 한다고 주장하면 두개의 바라지 않은 일들이 발생할것이다. 첫째로, 두개 또는 그이상의 다른 리상적인 보다 작은 모듈들이 일치적인 응집도를 가진 보다 큰 모듈을 만들기 위하여 하나의 덩어리로 합쳐 져야 한다. 둘째로, 관리하기에는 너무 크다고 생각되는 잘 설계된 모듈들에서 떼낸 일부 쪼각들이 다시 일치적인 응집도를 가진 모듈들로 된다.

일치적인 응집도가 그토록 나쁜 리유는 무엇인가? 일치적인 응집도를 가진 모듈들은 두가지 심중한 결함을 가지고 있다. 첫째로, 그러한 모듈들은 교정유지정비와 확장과 같은 제품의 유지정비성을 떨어 뜨린다. 제품을 파악하려고 하는 견지에서 보면 일치적인 응집도를 가진 모듈화는 모듈화가 전혀 되여 있지 않은것보다 더 나쁘다[Shneiderman and Mayer, 1975]. 둘째로, 이러한 모듈들은 재리용성이 없다. 이 절의 첫번째 단락에서 말한 일치적인 응집도를 가진 모듈은 임의의 다른 제품에서 전혀 다시 리용될수 있을것 같지 않다.

재리용성의 결핍은 심중한 결함이다. 쏘프트웨어를 만드는데 드는 비용이 너무 크기때문에 모듈을 가능한 모든 곳에서 다시 리용하기 위하여 노력하는것이 필수적이다. 어떤 모듈을 설계, 코드작성, 문서작성하며 더우기 시험하는것은 시간이 많이 들고 따라서 비용이 많이 드는 공정이다. 만일 잘 설계되고 철저히 시험되고 적당히 문서작성이 되여 있는 현존모듈이 또 다른 제품에서 리용될수 있다면 관리자측은 현존모듈이 재리용된다고 주장할것이다. 그러나 일치적인 응집도를 가진 모듈이 재리용될수 있는 방도가 전혀 없으면 그것을 개발하는데 소비된 비용은 절대로 보상될수 없다(재리용문제는 8장에서 상세히 론의한다.).

일반적으로 일치적인 응집도를 가진 모듈을 수정하는것은 쉽다. 왜냐하면 그것이 여러 작용을 수행하고 매개가 한가지 작용을 수행하는 보다 작은 모듈들로 쪼개지기 때문이다.

7. 2. 2. 론리적인 응집도

모듈은 그중 어느 하나가 호출하는 모듈에 의하여 선택되는 련관된 작용들을 련속 수행할 때 론리적인 응집도를 가지게 된다. 다음의 실례들은 모두 론리적인 응집도를 가진 모듈의 실례로 된다.

실례 1 다음과 같이 호출되는 모듈 new operation:

```
function code  = 7;
new operation (function code, dummy 1, dummy 2, dummy 3);
// dummy 1, dummy 2, dummy 3은 dummy 변수들이다.
// 만일 function code가 7과 같으면 리용되지 않는다.
```

이 실례에서 new operation은 네개의 인수들로 호출된다. 그러나 설명문에서 언급된 것처럼 만일 function code가 7과 같다면 그것들중 세개는 필요 없다. 이것은 그것이 정정이든 확장이든 유지정비에 대한 일반적인 의미에서 가독성을 떨어 뜨린다.

실례 2 모든 입출력을 실행하는 객체

실례 3 기본파일기록들에 대한 삽입, 삭제, 수정과 같은 편집기능을 수행하는 모듈

실례 4 OS/VS2의 초기의 판본에 있는 론리적인 응집도를 가진 모듈은 13개의 각이한 작용을 수행하였다. 그것의 대면부에는 21개의 자료토막들이 들어 있다[Myers, 1978b].

모듈이 론리적인 응집도를 가질 때 두가지 문제가 생긴다. 첫째로, 대면부를 리해하기 힘들다. 실례 1이 바로 그 경우에 해당된다. 전체적인 모듈에 대한 리해가능성은 그로 인하여 영향을 받을수 있다. 둘째로, 한가지이상의 작용에 대한 코드는 한데 얽히여 엄중한 유지정비문제들을 산생시킨다. 실례로 입출력을 모두 진행하는 모듈은 그림 7-5에서 보여 준것처럼 구조화될수 있다. 만일 새로운 테프장치가 설치되면 1, 2, 3, 4, 6, 9, 10이라고 번호를 붙인 부분들을 수정하는것이 필요할수도 있다. 이러한 변경들은 레이자인쇄기가 1과 3부분들에 대한 변경의 영향을 받기때문에 레이자인쇄기출력과 같은 다른 형태의 입출력에 부정적인 영향을 줄수 있다. 한데 얽혀 있는 특성은 론리적인 응집도를 가진 모듈들에서 특징적이다. 이 특성은 다른 제품들에서 그러한 모듈을 재리용하는것이 어렵다는 결과를 초래한다.

7. 2. 3. 림시적인 응집도

모듈은 련관된 일련의 작용들을 제때에 수행할 때 림시적인 응집도를 가진다. 림시적인 응집도를 가진 모듈의 한가지 실례는 **낡은 기본파일열기**와 **새로운 기본파일열기, 트랜잭션파일열기** 그리고 **파일인쇄, 판매지역표초기화, 첫 트랜잭션기록읽기와 첫 낡은 기본파일기록읽기**로 명명한 모듈이다. C/SD가 나오기전에는 그러한 모듈을 초기화실행이라고 불렀을것이다.

이 모듈의 작용들은 서로 약하게 련관되여 있지만 다른 모듈들의 작용들과는 더 강

하게 련관되여 있다. 실례로 **판매지역표**를 고찰해 보자. 그것은 이 모듈안에서 초기화되지만 **판매지역표갱신**과 **판매지역표인쇄**와 같은 작용은 다른 모듈들에 위치하고 있다. 따라서 만일 **판매지역표**의 구조가 변하게 되면 아마 그 기업체가 이전에 기업활동을 진행하지 못한 나라의 지역들로 활동을 확대하고 있기때문에 수많은 모듈들이 변하게 될것이다. 회귀오유(명백히 제품과 련관되여 있지 않은 부분에 생긴 변화로 하여 초래된 결함)가 보다 많이 생길 기회가 있을뿐아니라 만일 영향을 받은 모듈의 수가 크다면 한개 또는 두개의 모듈들이 무시되기 좋은 기회가 있다. 7.2.7에서 설명한것처럼 **판매지역표**에 대하여 진행되는 모든 조작들을 한개의 모듈안에 포함시키는것이 훨씬 더 좋다. 이 조작들은 필요할 때 다른 모듈들에 의하여 호출될수 있다. 이밖에 림시적인 응집도를 가진 모듈은 서로 다른 제품에서 재리용될수 없을것이다.

1.	모든 입력과 출력을 위한 코드
2.	입력만을 위한 코드
3.	출력만을 위한 코드
4.	디스크와 테프 I/O를 위한 코드
5.	디스크I/O를 위한 코드
6.	테프 I/O를 위한 코드
7.	디스크입력을 위한 코드
8.	디스크출력을 위한 코드
9.	테프입력을 위한 코드
10.	테프출력을 위한 코드
	⋮ ⋮ ⋮
37.	건반입력을 위한 코드

그림 7-5. 모든 입력과 출력을 실현하는 모듈

7. 2. 4. 절차적인 응집도

모듈은 만일 그것이 제품이 따르는 일련의 단계렬들에 의하여 관계되는 일련의 작용들을 수행한다면 절차적인 응집도를 가지게 된다. 절차적인 응집도를 가지고 있는 모듈의 실례는 **자료기지로부터 부분번호의 읽기**와 **유지정비파일상에서 수정기록의 갱신**이다.

이것은 명백히 림시적인 응집도보다 더 좋다. 즉 적어도 작용들은 절차적으로 호상련관되여 있다. 그렇다고 해도 동작들은 여전히 약하게 련관되여 있으며 또한 이 모듈은 다른 제품에서 재리용될것 같지 않다. 해결책은 절차적인 응집도를 가진 모듈을 독립적인 모듈들로 분해하여 매개 모듈이 한가지 작용을 수행하도록 하는것이다.

7. 2. 5. 통신적인 응집도

모듈은 만일 그것이 제품이 따르는 단계렬들에 의하여 관련되는 일련의 작용을 수행하고 모든 작용들이 동일한 자료상에서 수행된다면 통신적인 응집도를 가지게 된다. 통신적인 응집도를 가진 모듈의 두가지 실례로서 **자료기지에 있는 기록을 갱신하고 그것을 검사흔적에 쓰기, 새로운 자리길을 계산하고 그것을 인쇄기에 보내기**를 들수 있다. 이것은 모듈의 작용들이 보다 밀접하게 련관되여 있기때문에 절차적인 응집도보다는 더 좋다. 그러나 그것은 여전히 일치적인 응집도, 론리적인 응집도, 림시적인 응집도 그리고 절차적인 응집도와 똑 같은 결함을 가지고 있다. 즉 모듈이 재리용될수 없다는것이다. 그 해결책은 그러한 모듈을 독립적인 모듈들로 분해하여 매개 모듈들이 한가지 작용을 수행하도록 하는것이다.

겸사해서 베리(Berry)가 림시적인 응집도, 절차적인 응집도, 통신적인 응집도에 대하여 언급하면서 흐름도응집도(*flowchart cohesion*)라는 용어를 사용하고 있는것은 흥미 있다. 왜냐하면 그러한 모듈들에 의하여 수행되는 작용들이 제품흐름도에서 린접해 있기때문이다[Berry, 1978]. 림시적인 응집도의 경우에 작용들은 그것들이 동시에 수행되기때문에 린접해 있게 된다. 그것들은 절차적인 응집도에서는 알고리듬이 련속적으로 수행될 작용들을 요구하기때문에 린접해 있게 된다. 통신적인 응집도에서도 그것들을 련속적으로 수행되는것외에도 작용들이 똑같은 자료에 대하여 수행되기때문에 린접해 있게 된다. 따라서 이러한 작용들이 흐름도에서 린접해 있게 되는것은 응당하다.

7. 2. 6. 기능적인 응집도

하나의 동작을 정확히 수행하든가 혹은 단일한 목적을 실현하는 모듈은 기능적인 응집도를 가진다. 이와 같은 모듈의 실례에는 **용광로의 온도를 잰다; 전자의 자리길을 계산한다; 디스크에 쓴다; 판매수수료를 계산한다**가 있다.

기능적인 응집도를 가진 모듈은 그 모듈이 수행하는 하나의 작용이 다른 제품들에서 수행될 필요가 있기때문에 흔히 재리용될수 있다. 정확히 설계되고 철저히 시험되였으며 문서화가 잘된 기능적인 응집도를 가진 모듈은 임의의 쏘프트웨어기업체들에 있어서도 (경제적으로, 기술적으로) 가치가 있는것이며 될수록 자주 재리용되게 된다(7.5).

유지정비는 기능적인 응집도를 가진 모듈상에서 수행하기가 더 쉽다. 첫째로 기능적인 응집도는 오유를 고립시킨다. 만일 용광로의 온도가 정확히 읽어 지지 않는다는것이 사실이라면 오유는 거의나 확정적으로 모듈 **용광로의 온도를 잰다**에 있게 된다. 류사하게 만일 **전자의 자리길이 부정확하게 계산된다**면 처음으로 주시해야 할것은 **전자의 자리길을 계산한다**라는 모듈이다.

일단 오유가 단일한 모듈로 국부화되면 다음단계는 요구되는 변경을 진행하는것이다. 기능적인 응집도를 가진 모듈이 오직 하나의 작용을 수행하면 그러한 모듈은 일반적으로 보다 높은 응집도를 가진 모듈보다 리해하기가 더 쉽다. 이렇게 리해하기 쉬운것은 또한 유지정비를 간단하게 해준다. 마지막으로 변경이 진행될 때 특히 모듈들사이의 결합도가 작게 되면 변경이 다른 모듈들에 영향을 줄 기회는 적어 지게 된다(7.3).

기능적인 응집도는 또한 제품이 확장될 때 가치가 있다. 실례로 개인용콤퓨터가 10Mb의 하드구동기를 가지고 있는데 제작자는 이제 대신에 30Mb의 하드구동기를 가진 보다 강력한 콤퓨터의 모형을 시장에 내가려고 한다고 하자. 모듈의 목록을 읽고 나서 유지정비프로그람작성자는 **하드구동기에 쓴다**라는 모듈을 찾는다. 명백히 하여야 할것은 그 모듈을 **더 큰 하드구동기에 쓴다**라고 하는 새로운 모듈로 교체하는것이다.

겸사해서 다음과 같은 사실을 지적할 필요가 있다. 그림 7-2에 있는 세개의 모듈들은 기능적인 응집도를 가지며 그림 7-3의 설계보다 그림 7-2의 설계가 더 좋다는데 대하여 7.1에서 진행한 론증들은 정확히 기능적인 응집도가 더 좋다고 하는 앞의 론증과 같다.

7. 2. 7. 정보적인 응집도

모듈은 매개가 자기의 입력점을 가지고 있고 매 작용에 해당한 독자적인 코드가 있으며 모두 같은 자료구조에 기초하여 수행되는 수많은 동작들을 수행한다면 그것은 정보적인 응집도를 가지게 된다. 그 실례를 그림 7-6에서 보여 주고 있다. 이것은 구조화된 프로그람작성법에 어긋나지 않는다. 즉 매개 코드토막은 정확히 하나의 입력점과 하나의 출력점을 가지고 있다. 론리적인 응집도와 정보적인 응집도사이의 주되는 차이는 론리적인 응집도를 가진 모듈의 여러가지 동작들이 한데 얽혀 있다면 정보적인 응집도를 가진 모듈에서는 매개 동작의 코드가 완전히 독립적이라는것이다.

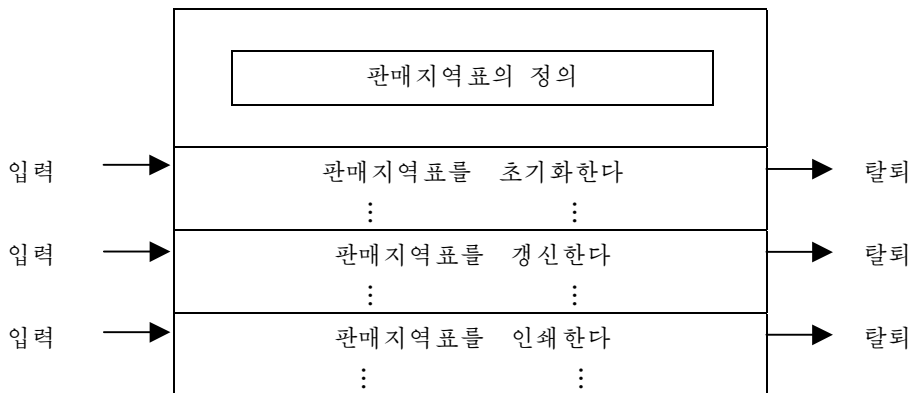



그림 7-6. 정보적인 응집도를 가진 모듈

정보적인 응집도를 가진 모듈은 7.5에서 설명한것처럼 본질상 추상자료형의 실현이며 추상자료형을 리용할 때의 모든 우월성은 정보적인 응집도를 가진 모듈이 리용될 때 얻어진다. 객체가 본질상 추상자료형(7.7)의 실례이기때문에 객체도 역시 정보적인 응집도를 가진 모듈이다. 그러므로 7.2에서는 정보적인 응집도가 객체지향파라다임[*)]에 대한 최량화

[*)] 이 단락에서 론의는 추상자료형이나 객체가 잘 설계된것을 전제로 하고 있다. 만일 대상의 방법이 전혀 관련이 없는 동작을 수행하게 된다면 객체는 일치적인 응집도를 가진다.

이라고 지적하였다.

7．2．8．응집도의 실례

응집도에 대하여 좀 더 심도 있게 고찰하기 위하여 그림 7-7에 있는 실례를 고찰해 보자. 특히 두개의 모듈을 설명할 필요가 있다. **합을 초기화하고 파일열기**와 **파일들을 닫고 평균온도를 인쇄**라는 모듈들이 모두 림시적인 응집도가 아니라 일치적인 응집도를 가진다고 명시되여 있다는 사실은 어느 정도 놀라울수 있다. 우선 모듈**합을 초기화하고 파일열기**를 생각해 보자.

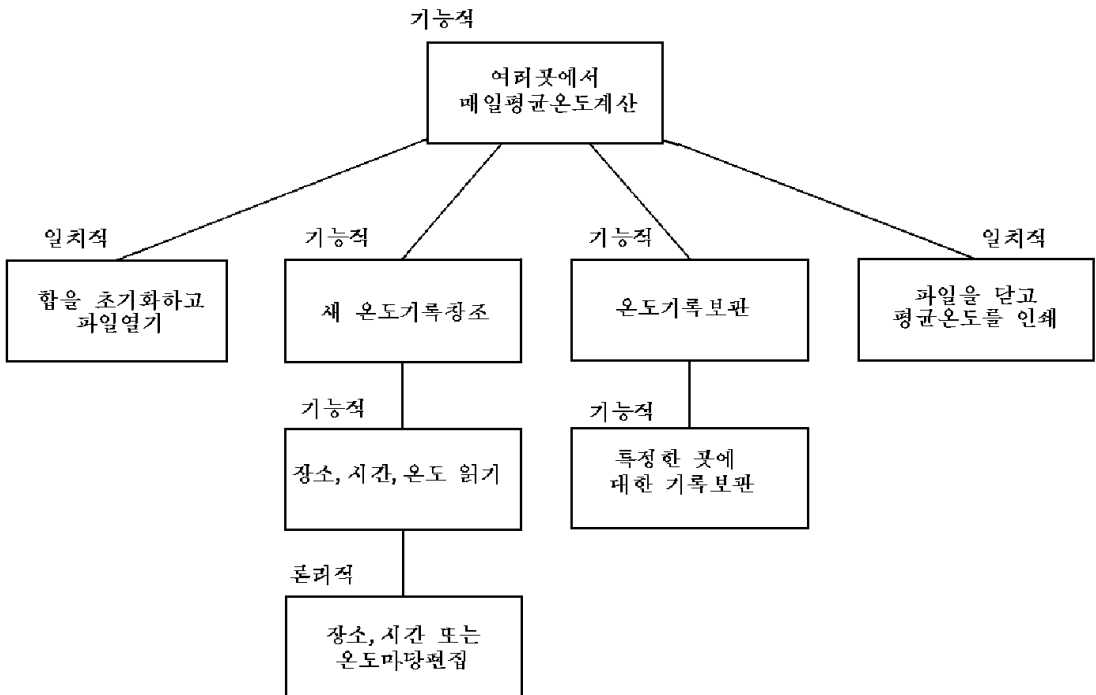



그림 7-7. 매개 모듈의 응집을 보여 주는 모듈호상접속도

그것은 두 작용들이 임의의 계산들을 수행하기전에 수행되여야 한다는 점에서 제때에 관련되는 그 두개의 작용들을 수행한다. 따라서 모듈이 림시적인 응집도를 가지고 있는것처럼 보인다. 비록 **합을 초기화하고 파일열기**의 두 작용이 정말로 계산의 초기에 수행된다고 해도 또 다른 요인이 포함된다. 합을 초기화하는것은 문제와 관련이 있지만 파일열기는 문제 그자체와는 아무런 관련이 없는 하드웨어적인 문제이다. 두개 혹은 그이상의 서로 다른 준위의 응집도들이 하나의 모듈에 할당할수 있을 때의 규칙은 가장 낮은 가능한 준위를 할당하는것이다. 따라서 **합을 초기화하고 파일열기**가 림시적인 응집도이든가 일치적인 응집도를 가질수 있기때문에 두 준위의 응집도들중 보다 더 낮은 일치적인 응집도가 그 모듈에 할당되게 된다. 이것은 또한 **파일들을 닫고 평균온도를 인쇄**가 일치적

인 응집도를 가지게 되는 리유로 된다.

7. 3. 결 합 도

응집도가 어떤 모듈내에서의 호상작용의 정도라는것을 상기해 보자. 결합도는 두 모듈사이의 호상작용의 정도이다. 응집도는 그림 7-8에서 보여 주는바와 같이 여러가지 준위들로 구분할수 있다. 좋은 결합을 강조하기 위하여 여러가지 준위들을 가장 나쁜것부터 가장 좋은것의 순서로 설명하기로 한다.

5.	자료결합(좋다)
4.	스탬프결합
3.	조종결합
2.	공통결합
1.	내용결합(나쁘다)

그림 7-8. 결합의 준위

7. 3. 1. 내용결합

두 모듈은 만일 한개의 모듈이 다른 모듈의 내용들을 직접적으로 참고한다면 내용상 결합되여 있다. 다음의것들은 모두 내용결합의 실례들이다.

실례 1 모듈 p는 모듈 q의 명령문을 수식한다.

이러한 실천은 아쎔블리어프로그람작성에 국한되여 있지 않다. 현재는 COBOL에서 제거된 동사 **alter**는 정확히 또 다른 명령문을 수식한다.

실례 2 모듈 p는 모듈 q내에서 몇개의 수자적인 치환에 의하여 모듈 q의 국부자료를 참조한다.

실례 3 모듈 p는 모듈 q의 국부적인 표식으로 분기한다.

모듈 p와 모듈 q가 내용적으로 결합되여 있다고 가정하자. 여러가지 위험들중의 하나는 모듈 q에 생긴 임의의 변경은 q가 새로운 콤파일러 혹은 아쎔블리어에 의하여 재콤파일된다고 하여도 모듈 p의 변경을 요구한다는것이다. 더우기 모듈 q를 재리용함이 없이 어떤 새로운 제품에서 모듈 p를 재리용한다는것은 불가능하다. 두 모듈이 내용결합될때 그것들은 뗄수없이 련결된다.

7. 3. 2. 공통결합

만일 두 모듈들이 동일한 대역적인 자료에 접근한다면 그 두 모듈은 공통결합된다. 그에 대하여 그림 7-9에 보여 주었다. 인수들을 넘겨 주어 서로 자료통신을 진행할 대신에 모듈 cca와 ccb는 대역변수(*global variable*)의 값을 호출하여 변경시킬수 있다. 이것이 일어 날수 있는 가장 일반적인 상황은 모듈 cca와 모듈 ccb가 모두 같은 자료기지에 접근하고 또 동일한 기록에 대하여 읽기쓰기를 진행할수 있는 경우이다. 공통결합에 있어서는 두 모듈이 자료기지에 대한 읽기쓰기를 할수 있어야 한다. 만일 자료기지접근방식이 읽기전용방식이라면 공통결합방식이 아니다. 그러나 C++나 Java의 수식어 **public**의 리용을 비롯하여 공통결합을 실현하는 다른 방법들도 있다.

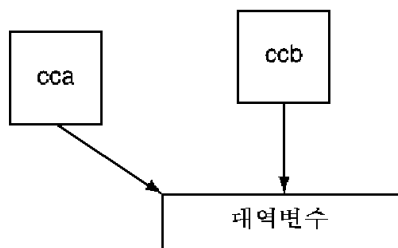



그림 7-9. 공통결합

이러한 결합형태는 많은 리유로 하여 좋은것이 못된다. 그것은 첫째로, 결과코드를 사실상 읽을수 없기때문에 구조화프로그람작성과 모순된다. 그림 7-10에 있는 의사코드 토막을 생각해 보자.

```
while (global variable = = 0)
 {
   if (argument xyz>25)
      module 3 ();
   else
      module 4 ();
 }
```

그림 7-10. 공통결합을 반영한 의사코드

만일 global variable이 대역변수이라면 그 값은 module 3, module 4 혹은 그것들에 의하여 호출되는 임의의 모듈에 의하여 변경될수 있다. 어떤 조건에서 순환이 종결되는가를 결정하는것은 자명한 문제가 아니다. 즉 만일 실시간오유가 발생한다면 수많은 모듈들가운데서 임의의 한 모듈이 global variable의 값을 변경시켰을수 있기때문에 발생한 오유를 수정하는것은 어려울것이다.

둘째로, 모듈들이 그것들에 대한 읽기가능성에 영향을 주는 부작용을 가지고 있을수 있다. 실례로 edit this transaction (record 7)을 생각해 보자. 만일 공통결합이 있다면 이 호출은 바로 record 7의 값뿐만아니라 그 모듈에 의하여 접근될수 있는 임의의 대역변수를 변경시킬수 있다. 한마디로 전체 모듈이 정확히 그것이 무엇을 하는가를 밝혀 내기 위하여 읽어 져야 한다.

셋째로, 만일 한 모듈에서 대역변수의 선언에 대하여 유지정비변화가 진행된다면 그 대역변수에 접근할수 있는 매개 모듈이 변경되여야 한다. 더우기 모든 변경들은 모순이 없어야 한다.

넷째로, 모듈이 재리용될 때마다 대역변수들의 동일한 목록이 제공되여야 하기때문에 공통결합된 모듈은 재리용하기 힘든것이다.

다섯째로, 잠재적으로 가장 위험한 문제로서 공통결합의 결과로 어떤 모듈은 그것이 요구하는것보다 더 많은 자료를 로출시킬수 있게 된다. 이것은 자료접근을 조종하려는 그 어떤 시도도 좌절시키며 극단한 경우에는 콤퓨터범죄를 초래할수 있다. 많은 형태의 콤퓨터범죄들은 일정한 형태의 공모결탁을 필요로 한다. 잘 설계된 쏘프트웨어는 그 어떤 프로그람작성자도 범죄를 범하는데 필요한 모든 자료나 모듈들에 접근하지 못하도록 하여야 한다. 실례로 로임지불계산제품의 검열인쇄부분을 작성하는 프로그람작성자는 종업원기록들에 접근할 필요가 있다. 그러나 잘 설계된 제품에서 그러한 접근은 배타적으로 읽기전용방식으로 진행되며 따라서 프로그람작성자가 자기의 월로임을 승인없이 변경할수 없게 있다. 그러한 변경을 하자면 프로그람작성자는 갱신방식으로 련관된 기록들에 접근할수 있는 또 다른 정직하지 못한 종업원에 의거하여야 한다. 그러나 만일 제품이 잘 설계되지 못하고 모든 모듈이 갱신방식으로 된 로임지불명부자료기지에 접근할수 있다면 단독으로 행동하는 비도덕적인 프로그람작성자가 자료기지에 있는 임의의 기록을 승인없이 변경할수 있다.

비록 앞에서 렬거한 리유들이 대다수의 대담한 독자들을 제외한 모든 사람들이 공통결합을 리용하지 못하게 할것을 요구한다고 할지라도 공통결합이 개별적인 선택보다 더 좋은 경우들이 존재한다. 실례로 원유저장탕크들에 대한 콤퓨터지원설계를 수행하는 제품에 대하여 생각해 보자[Schach and Stevens-Guille, 1979]. 탕크는 높이, 직경, 탕크가 받게 될 최대바람속도, 절연두께와 같은 수많은 서술자들로 렬거되게 된다. 서술자들은 초기화되여야 하지만 그에 따라 값에서는 변경되지 말아야 한다. 그리고 제품에 있는 대부분의 모듈들은 서술자들의 값에 접근할 필요가 있다. 55개의 탕크서술자가 있다고 가정하자. 만일 이 모든 서술자들이 매개의 모듈에 인수로서 넘어 간다면 매개 모듈의 대면부는 적어도 55개의 인수들로 구성될것이며 오유를 범할 가능성은 크다. 인수들에 대한 엄격한 검사를 요구하는 Ada와 같은 언어들에서도 동일한 류형을 가진 두 인수들은 여전히 호상 교체될수 있으며 형검사기에 의하여 검출되지 못하는 오유가 있을수 있다.

한가지 해결책은 모든 탕크서술자를 자료기지에 넣는것이며 한 모듈이 모든 서술자의 값들을 초기화하고 반면에 다른 모든 모듈이 읽기전용방식으로 배타적으로 자료기지에 접근하는 방식으로 제품을 설계하는것이다. 그러나 만일 규정된 실현언어가 유용한 자료기지관리체계와 대화할수 없는것으로 하여 자료기지해결방안이 실천적인 방도로 되

지 않는다면 하나의 대안은 조종방식으로서만 공통결합을 리용하는것이다. 즉 제품은 55개의 서술자가 한개의 모듈에 의하여 초기화되지만 그 어떤 다른 모듈도 서술자의 값을 변경시키지 않도록 설계되여야 한다. 이러한 프로그람작성형식은 쏘프트웨어에 의하여 시행되는 자료기지해결방안과는 달리 관리자에 의하여 시행되여야 한다. 따라서 공통결합의 리용과 관련한 그 어떤 좋은 다른 방도가 없는 경우에 대하여서는 관리자에 의한 세밀한 감독에 의하여 일련의 위험들을 줄일수 있다. 그러나 보다 좋은 해결책은 7.6에 설명되는바와 같이 정보은페를 리용하여 공통결합을 미리 방지하는것이다.

7. 3. 3. 조종결합

만일 한 모듈이 조종요소를 다른 모듈에 넘긴다면 그 두 모듈들은 조종결합된다. 즉 한 모듈은 명백히 다른 모듈의 론리를 조종한다. 실례로 한 기능코드가 론리적인 응집도를 가진 어떤 모듈에 넘겨 질 때 조종이 넘겨 진다(7.2.2). 조종결합의 또 다른 설계는 조종스위치가 하나의 인수로서 넘겨 지는 경우이다.

만일 모듈 p가 모듈 q를 호출하고 모듈 q가 모듈 p에 《나는 나의 과제를 끝낼수 없다》라는 표식을 되돌려 넘겨 주게 되면 모듈 q는 자료를 넘겨 준다.

그러나 만일 표식이 《나는 나의 과제를 끝낼수 없다. 따라서 오유통보문 **ABC123**를 써넣으라.》라는것을 의미한다면 모듈 p와 q는 조종결합된다. 달리말하여 만일 모듈 q가 정보를 모듈 p에 되돌려 넘겨 주고 모듈 p가 그 정보를 받은 결과로서 어떤 동작을 취해야 할것인가를 결정한다면 모듈 q는 자료를 넘겨 준다.

조종결합에 의하여 발생하는 기본난관은 두 모듈들이 독립이 아니라는것이다. 즉 호출된 모듈 q는 모듈 p의 내부구조와 론리를 알고 있어야 한다. 결과 재리용가능성은 줄어 든다. 이밖에 조종결합은 일반적으로 론리적인 응집도를 가진 모듈과 련관되는데 론리적인 응집도와 관련하여 난관들이 존재하게 된다.

7. 3. 4. 스탬프결합

일부 프로그람작성언어들에서는 part number, satellite altitude 혹은 degree of multiprogramming와 같은 단순한 변수들만이 인수들로서 넘겨 질수 있다. 그러나 많은 언어들은 또한 기록들이나 배렬들과 같은 자료구조들이 인수들로서 넘어 가는것을 지원한다. 이와 같은 언어들에서 타당한 인수들에는 part record, satellite coordinates 혹은 segment table이 포함된다. 만일 하나의 자료구조가 인수로 넘겨 지지만 호출된 모듈이 그 자료구조의 일부 개별적요소들에 대해서만 작용한다면 이 두개의 모듈들은 스탬프결합된다.

실례로 호출 calculate withholding**(**employee record**)**를 생각해 보자. 전체 calculate withholding모듈을 읽지 않고서는 모듈이 employee record의 어느 마당에 접근하는가 또는 어느 마당을 변경시키는가 하는것이 명백치 않다. 종업원들의 로임을 명백하게 넘겨 주는것은 공제액을 계산하는데서 필수적이지만 종업원의 집전화번호가 이 목적에 어떻게

필요되는가 하는것을 아는것은 힘들다. 그대신 공제액을 계산하기 위하여 실제적으로 필요한 마당들만이 모듈 calcuate withholding에 넘어 가야 한다. 결과적인 모듈 특히 그의 대면부는 더 리해하기 쉬울뿐만아니라 공제액을 계산하는데 필요한 여러가지 다른 제품들에서 재리용할수 있을것 같다(이에 대한 다른 내용을 알자면 다음의 《알고 싶은 문제》를 보시오).

알고 싶은 문제

4개 혹은 5개의 각이한 마당들을 하나의 모듈에 넘기는것은 하나의 완전한 기[illegible]을 넘기는것보다 더 시간이 걸릴수 있다. 이 상황은 하나의 보다 큰 문제를 초래한다. [illegible](응답시간 혹은 공간제한과 같은) 최량화문제들이 일반적으로 훌륭한 쏘프트웨어공학실천이라고 간주되는것과 모순될 때 무엇을 해야 하는가 하는 문제이다.

경험에 의하면 이러한 질문은 흔히 적합치 않는것으로 판명된다. 권고된 방법[illegible] 응답시간을 느리게 할수 있지만 이것은 불과 1ms정도로 너무 작아서 사용자들이 알[illegible] 낼수 없다. 그러므로 돈 크누스(Don Knuth)의 최량화제1법칙 즉 《하지 말라!》에 따르[illegible] 성능상 리유를 비롯하여 임의의 종류의 최량화에 대한 요구는 거의 없다 [Knuth, 1974].

그러나 최량화가 실제로 요구된다면 어떻게 하겠는가? 이 경우에 크누스의 최[illegible]화제2법칙이 적용된다. 제2법칙(전문가들만이라고 표제를 단)은 다음과 같다. 《아직[illegible] 아니다!》 달리 말하면 우선 적당한 쏘프트웨어공학기법을 리용하여 전체적인 제품을 완성한다. 그다음 만일 최량화가 실제적으로 요구된다면 무엇이 무엇때문에 변경되고 있는가를 세밀히 문서작성하면서 필요한 변경만 진행한다. 만일 조금이라도 가능하다면 이[illegible] 한 최량화는 경험 있는 쏘프트웨어공학자에 의하여 진행되여야 한다.

호출 calculate withholding (employee record)가 엄격히 필요한것보다 더 많은 자료를 넘겨 주기때문에 아마 훨씬 더 중요한 문제로서 조종되지 않은 자료호출문제들 그리고 생각컨데 콤퓨터범죄가 다시금 생길수 있다. 이 문제는 7.3.2에서 론의되였다.

만일 자료구조의 모든 요소들이 호출된 모듈에 의하여 리용된다는 조건하에서 하나의 자료구조를 인수로서 넘기는것은 아무런 오유도 없다. 실례로 invert matrix (original matrix, inverted matrix) 또는 print inventory record (warehouse record)와 같은 호출들은 자료기지를 인수로 넘기지만 호출된 모듈들은 그 자료구조의 모든 요소들에 작용한다. 자료구조가 인수로 넘겨 질 때 스탬프결합이 존재하지만 일부 요소들만이 호출된 모듈에서 리용된다.

하나의 미묘한 형식의 스탬프결합은 C 혹은 C++와 같은 언어들에서 하나의 레코드에 대한 지적자가 인수로서 넘겨 지는 경우에 발생할수 있다. 호출 check altitude (pointer to position record)를 고찰하자. 척 보기에는 넘겨 지는것은 하나의 단순한 변수이다. 그러나 호출된 모듈은 pointer to position record에 의하여 지적된 position record의 모든 항목들에 접근한다. 잠재적인 문제들로 인하여 지적자가 인수로 넘겨 질 때마다 결합을 세밀히 조사해 보는것이 좋다.

7. 3. 5. 자료결합

만일 모든 인수들이 같은 종류의 자료항목들이라면 두 모듈은 자료결합된다. 즉 매 인수는 단순한 인수이거나 모든 요소들이 호출된 모듈에 의하여 리용되는 하나의 자료구조이다. 그 실례들로는 display time of arrival(flight number), compute product(first number, second number, result) 그리고 determine job with highest priority(job queue)들이 있다.

자료결합은 하나의 바람직한 목표이다. 그것을 부정적인 방법으로 실현하기 위하여 만일 어떤 제품이 오직 자료결합을 나타낸다면 내용, 공통, 조종 그리고 스탬프결합에서의 난관들은 생기지 않을것이다. 보다 더 긍정적인 견지에서 보면 만일 두 모듈이 자료결합된다면 하나의 모듈에 대한 변경이 다른 모듈에서의 회귀오유를 적게 발생시킬수 있기때문에 유지정비가 더 쉽다. 다음의 실례들은 결합의 일정한 측면들을 보여 준다.

7. 3. 6. 결합의 실례

그림 7-11에 있는 실례를 고찰해 보자. 릉에 있는 수자들은 그림 7-12에서 보다 더 상세히 정의되는 대면부들을 나타낸다. 실례로 모듈 p가 모듈 q(대면부 1)를 호출할 때 그것은 하나의 인수 즉 비행기류형을 넘긴다. 모듈 q가 모듈 p에 조종을 넘길 때 그것은 상태기발표식을 되돌려 넘겨 준다. 그림 7-11과 7-12의 정보들을 리용하여 매 모듈쌍들 사이의 결합이 추론될수 있다. 그 결과를 그림 7-13에 보여 주었다.

그림 7-13에 있는 일부 입력점들은 명백하다. 실례로 p와 q(그림 7-11에 있는 대면부 1)사이 그리고 r와 t(대면부 5)사이, s와 u(대면부 6)사이의 자료결합은 하나의 단순한 변수가 매개 방향으로 넘겨 진다는 사실에 대한 직접적인 결과이다. 만일 p로부터 s로 넘겨 진 부분품목록에 있는 모든 요소들이 리용되거나 갱신되면 p와 s(대면부 2)사이의 결합은 자료결합으로 될수 있다. 그러나 만일 s가 목록의 일정한 요소들에만 작용한다면 그것은 스탬프결합이다. q와 s(대면부 4)사이의 결합은 이와 비슷하다. 그림 7-11과 7-12의 정보가 여러가지 모듈들의 기능을 완전히 설명하지 못하고 있기때문에 결합이 자료결합인가, 스탬프결합인가 하는것을 결정할 방도는 없다. q와 r(대면부 3)사이의 결합은 기능코드가 q로부터 r에로 넘겨 지기때문에 조종결합이다.

아마 좀 놀라운것은 그림 7-13에 공통결합이라고 표시된 세개의 입력점들이다. 그림 7-11에서 제일 멀리 떨어 져 있는 세개의 모듈쌍들 즉 p와 t, p와 u, t와 u는 처음에는 그 어떤 방향으로 결합되지 않는것 같이 보인다. 결국 그것들을 련결하는 대면부가 없다. 그리하여 공통결합은 물론 그것들사이의 결합에 대한 착상은 일정한 설명을 요구한다. 그 대답은 그림 7-11의 오른쪽에 있는 주해 즉 p, t, u 가 갱신방식으로 동일한 같은 자료기지에 접근한다는 주해에 있다. 그 결과는 수많은 대역변수들이 세개의 모듈들에 의하여 변화될수 있으며 그것들은 쌍방향으로 공통결합한다는것이다.

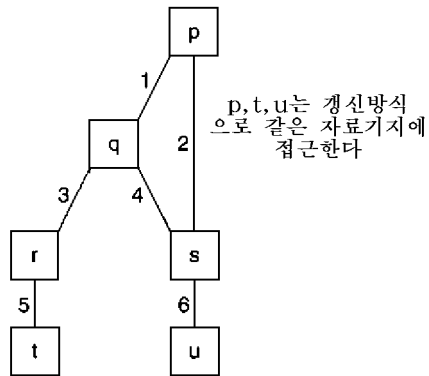



그림 7-11. 결합실례를 보여 주는 모듈호상접속도

Number	In	Out
1	비행기류형	상태기발
2	비행기의 부분품목록	-
3	기능코드	-
4	비행기의 부분품목록	-
5	부분품번호	부분품제작자
6	부분품번호	부분품이름

그림 7-12. 그림 7-11의 대면부서술

	q	r	S	t	u
p	자료	–	자료 또는 스탬프	공통	공통
q		조종	자료 또는 스탬프	–	–
r			–	자료	–
s				–	자료
t					공통

그림 7-13. 그림 7-11의 모듈들사이의 결합

7. 3. 7. 결합의 중요성

결합도는 중요한 척도이다. 만일 모듈 p가 모듈 q에 단단히 결합된다면 모듈 p에 대한 어떠한 변경은 모듈 q에 대한 대응하는 변경을 요구할수 있다. 만일 이 변경이 통합단계나 유지정비단계에서 요구한대로 진행된다면 결과적인 제품은 기능을 정확히 수행할것이다. 그러나 그 단계에서의 전진속도는 결합이 보다 더 느슨해 진 경우에 진행된것보다 더 떠질것이다. 다른 한편 만일 그때 필요한 변경이 모듈 q에서 진행되지 않는다면 그후에 오유가 저절로 나타나게 될것이다. 가장 좋은 경우에 콤파일러나 련결프로그람은 팀에 무엇인가 오유가 생기거나 모듈 p에 대한 변경을 시험하는 동안에 오유가 발생할것이라는것을 즉시에 통보해 줄것이다. 그러나 보통 일어 나는것은 다음에 진행되는 통합시험이나 제품이 의뢰자의 콤퓨터에 설치된후에 제품이 실패한다는것이다. 두 경우들에서는 모듈 p에 대한 변경이 끝난 다음에 오유가 발생한다. 모듈 p에 대한 변경과 모듈 q에 대한 스쳐 버린 해당한 변경들사이에는 아무런 명백한 련결도 없다. 따라서 오유를 발견하기 어려울수 있다.

모듈들이 높은 응집도와 낮은 결합도를 가지게 되는 설계가 훌륭한 설계라고 하면 명백히 다음과 같은 질문이 제기된다. 즉 그러한 설계를 어떻게 실현할수 있는가? 이 장은 설계와 관련한 리론적인 개념들을 서술하기때문에 이 문제에 대한 대답은 13장에 제시한다. 이 과정에 좋은 설계를 확증해 주는 품질들을 더 검토하고 세련시키게 된다. 편의상 이 장에서 기본적인 정의들은 그림 7-14에서 매개 정의들이 지적되여 있는 절들과 함께 제시된다.

추상자료형	그 자료형의 실체에 대하여 수행하는 동작을 포함한 자료형 (7.5)
추상화	불필요한 세부는 삭제하고 관련 있는 세부는 강조하여 계단식으로 세련하는 방법(7.4.1)
클라스	계승을 지원하는 추상자료형 (7.7)
응집도	모듈내에서의 호상작용정도 (7.1)
결합도	두 모듈사이의 호상작용정도 (7.1)
자료교감화	그 자료구조에 대하여 수행하는 작용을 포함한 자료구조 (7.4)
교감화	그 단위에 의하여 모형화한 모든 측면의 실세계의 실체에 대하여 하나의 단위로 모아 놓은것(7.5)
정보은페	결과적인 실현세부가 다른 모듈로부터 숨겨 지도록 설계를 구조화하는것 (7.6)
객체	클라스의 실체(7.7)

그림 7-14. 이 장의 기본정의와 해당한 절

7. 4. 자료의 교갑화

대형콤퓨터용조작체계를 설계하는 문제를 생각해 보자. 명세서에 따라 콤퓨터에 부과된 일감은 우선도가 높은 일감, 중간인 일감, 우선도가 낮은 일감으로 분류된다. 조작체계의 과제는 다음에 어느 일감을 기억장치에 넣어 줄것인가 하는것을 결정하는것이다. 즉 기억장치의 일감들중 어느것이 다음 시간토막을 차지하며 또 그 시간토막이 얼마나 지속되는가 그리고 디스크호출을 요구하는 일감들중 어느것이 우선도가 가장 높은가 하는것들을 결정하는것이다. 이러한 처리계획작성을 진행하는데서 조작체계는 매 일감의 우선도를 고려해야 한다. 즉 우선도가 높을수록 그 일감에 콤퓨터의 자원들이 더 빨리 할당되여야 한다. 이것을 실현하는 한가지 방도는 매개 일감우선도(*job-priority*)준위에 해당한 개별적인 일감대기렬(*job queue*)을 유지하는것이다. 일감대기렬은 초기화되여야 한다. 그리고 조작체계가 그 일감에 필요한 자원을 할당할것을 결정할 때 대기렬로부터 일감을 제거할뿐아니라 일감이 기억, CPU시간 혹은 디스크호출을 요구할 때 일감을 일감대기렬에 추가하기 위한 기능이 있어야 한다.

간단한 문제로서 기억호출을 위하여 대기렬로 작성된 묶음일감들(*batch jobs*)에 대한 제한된 문제를 생각해 보자. 련이은 묶음일감을 위한 세개의 대기렬이 있는데 매 우선도준위에 하나의 일감이 할당된다. 일감은 사용자가 제기할 때 적당한 대기렬에 추가되며 조작체계가 일감이 실행될 준비가 되였다고 결정할 때 대기렬로부터 제거되고 기억령역이 거기에 할당된다.

제품에서 이 부분은 여러가지 각이한 방식으로 구축할수 있다. 그림 7-15에서 보여 준 한가지 설계는 세개의 일감대기렬중 하나를 처리하기 위한 모듈들을 서술해 주고 있다. 의사코드와 같이 C언어는 고전적인 파라다임에서 제기될수 있는 일부 문제들을 강조하는데 리용된다. 이 장의 뒤부분에서 이 문제들은 객체지향파라다임을 리용하여 해결하고 있다.

그림 7-15를 고찰해 보자. 모듈 **m_1**에서 함수 **initialize_job_queue**[*)]는 일감대기렬의 초기화에 리용된다. 그리고 모듈 **m_2**와 **m_3**에 있는 함수 **add_job_to_queue**와 **remove_job_from_queue**들은 각각 일감의 추가와 삭제에 리용된다. 모듈 **m_123**에는 이 일감대기렬을 조작하기 위하여 세개의 모든 함수들에 대한 호출들을 포함한다. 자료교갑화에 집중하기 위하여 자리부족(*underflow*)(빈대기렬로부터 일감을 제거하기 위한 시도)과 자리넘침(*overflow*)(가득찬 대기렬에 일감을 추가하기 위한 시도)과 같은 문제들은 이 장의 나머지 부분에서와 마찬가지로 여기서는 밝히지 않는다.

그림 7-15의 설계에 있는 모듈들은 일감대기렬에 대한 작용들이 제품전반에 걸쳐 분산되여 있기때문에 낮은 응집도를 가진다. 만일 **job_queue**가 실현되는 방식 (실례로 선형의 목록대신 기록들의 련결된 목록과 같이)을 변경하기 위한 결심을 내린다면 모듈 **m_1, m_2, m_3**은 철저히 수정되여야 한다. 모듈 **m_123**도 역시 변경되여야 한다.

[*)] 보다 명백히 하기 위하여 이 에서는 구조화파라다임이 리용된다는것을 강조하기 위하여 **initialize_job_queue**와 같이 함수이름에 밑줄표식을 달았다. 다음 절에서 대상지향파라다임을 리용할 때 그에 해당한 방법을 **initializeJobQueue**라고 쓴다.

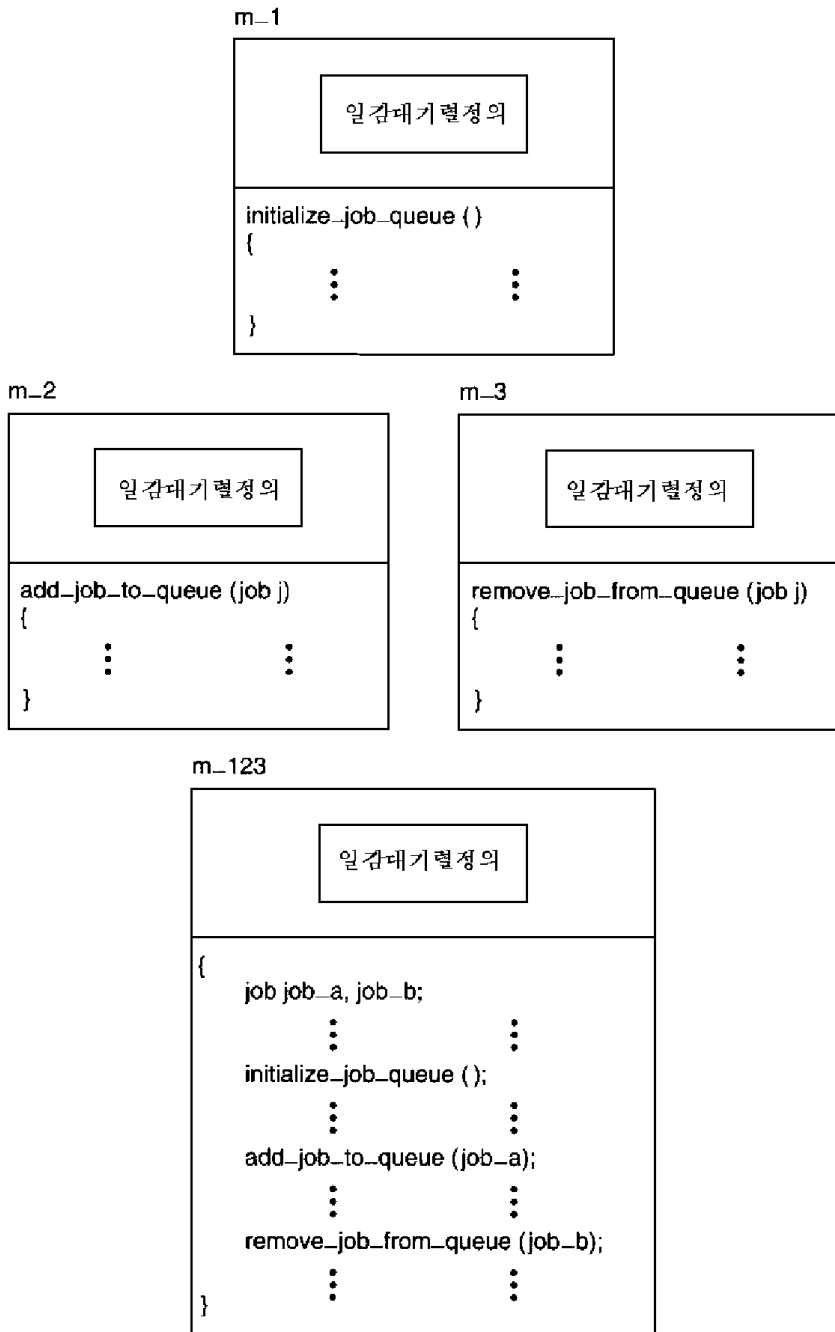



그림 7-15. 조작체계의 일감대기렬부분의 한가지 설계

이제 그림 7-16의 설계가 대신 선택되였다고 하자. 그림의 오른쪽에 있는 모듈은 그것이 동일한 자료구조에 대하여 여러가지 작용들을 수행한다는 점에서 정보적인 응집도를 가진다(7.2.7). 매 작용은 고유한 입력점과 탈퇴점 그리고 독자적인 코드를 가지고 있

다. 그림 7-16에서 모듈 **m_encapsulation**은 자료교갑화 즉 그 자료구조에서 수행될 작용들을 포함한 자료구조 이 경우에는 그러한 일감대기렬의 실현으로 된다.

```
m_123
{
    job  job_a, job_b;
        ⋮          ⋮
    initialize_job_queue ( );
        ⋮          ⋮
    add_job_to_queue (job_a);
        ⋮          ⋮
    remove_job_from_queue (job_b);
        ⋮          ⋮
}
```

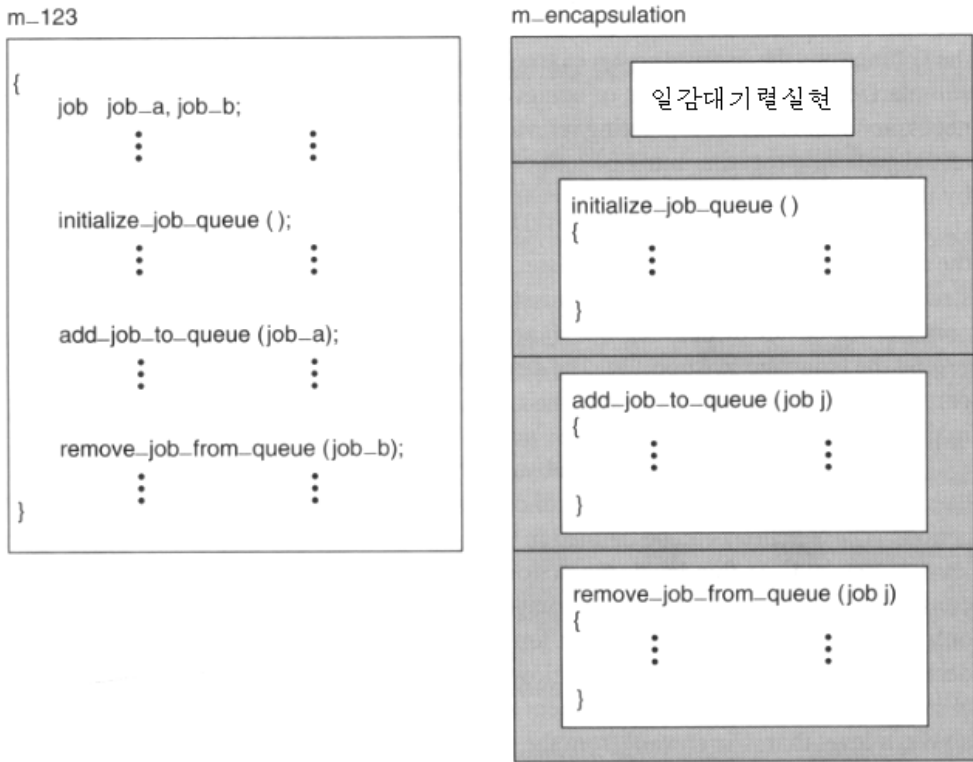



그림 7-16. 자료교갑화를 리용한 조작체계의 일감대기렬부분의 설계

이 시점에서 하나의 명백한 문제는 자료교갑화를 리용하여 제품을 설계하는 우점은 무엇인가 하는것이다. 이 질문에 대하여서는 두가지 방법 즉 개발의 견지에서와 유지정비의 견지에서 대답을 할수 있다.

7. 4. 1. 자료교갑화와 제품개발

자료교갑화는 추상화의 실례이다. 일감대기렬실례로 되돌아 가면 자료구조(일감대기렬)는 세개의 련관된 작용들(일감대기렬초기화, 대기렬에 일감을 추가, 대기렬에서 일감을 제거)과 함께 정의되였다. 개발자는 기록들이나 배렬들의 보다 낮은 준위에서가 아니라 일감들과 일감대기렬의 보다 높은 준위에서 문제를 개념화할수 있다.

추상적개념의 리면에 있는 기본 리론적인 개념은 계단식세련이다. 우선 제품에 대한 설계는 일감들, 일감대기렬들 그리고 일감대기렬들에서 수행되는 작용들과 같은 높은 준위의 개념들에 의하여 진행된다. 이 단계에서 일감대기렬이 어떻게 실현되는가 하는것은

완전히 무의미하다. 일단 완전한 높은 준위의 설계가 얻어 지면 두번째 단계는 자료구조와 그 자료구조에 대하여 수행되는 작용들이 실현될 보다 낮은 준위의 구성요소들을 설계하는것이다. 실례로 C언어에서 자료구조(일감대기렬)는 기록들(구조들) 혹은 배렬들에 의하여 실현될것이다. 세개의 작용(일감대기렬초기화, 대기렬에 일감을 추가, 대기렬에서 일감을 제거)들이 함수들로서 실현될것이다. 기본문제점은 이보다 더 낮은 준위를 설계할 때 개발자가 일감들, 일감대기렬들 그리고 작용들에 대한 목적한 리용을 완전히 무시해 버리는것이다. 따라서 첫번재 단계에서 비록 그 단계에서는 해당한 준위에 대하여 그 어떤 고려도 하지 않는다고 하여도 보다 낮은 준위가 존재한다는것을 가정한다. 그리고 두번째 단계(보다 낮은 준위의 설계)에서는 보다 높은 준위의 존재를 무시한다. 보다 더 높은 준위에서는 자료구조와 일감대기렬의 거동에 관심을 가지게 되며 보다 낮은 준위에서는 그 거동의 실현이 1차적인 관심사로 된다. 물론 보다 큰 제품은 많은 추상준위를 가지게 될것이다.

각이한 형태의 추상화가 존재한다. 그림 7-16을 생각해 보자. 그림에는 두가지 형태의 추상화가 있다. 자료교갑화 즉 자료구조와 그 자료구조에 대하여 수행하게 될 작용은 자료추상화의 실례로 된다. C언어의 함수 그자체는 처리절차추상화의 실례로 된다. 추상화는 개괄하면 단순히 불필요한 세부적인것을 없애고 관계되는 세부적인것들을 강조함으로써 계단식세련을 실현하는 수단이다. 자료교갑화는 모형화될 실세계의 실체에 대한 모든 측면들을 하나의 단위로 종합하는것으로 정의할수 있다. 이것을 1.6에서는 개념적독립성이라고 하였다.

자료추상화는 설계가가 자료구조와 자료구조에 대하여 수행되는 작용들의 준위를 고찰하고 그 다음에 자료구조와 작용들이 어떻게 실현되는가 하는 세부에 집중할수 있도록 한다. 이제는 처리절차추상화로 방향을 바꾸어 C언어함수 **initialize_job_queue**의 정의에 대한 결과를 고찰하자. 이 효과는 개발자에게 다른 기능 즉 원래 정의된 언어의 부분이 아닌 기능을 제공함으로써 그 언어를 확장하는것이다. 개발자는 **sqrt** 혹은 **abs**와 같은 방식으로 **initialize_job_queue**를 리용할수 있다.

설계를 위한 처리절차추상화의 의미는 자료추상화의 의미와 마찬가지로 위력하다. 설계자는 높은 준위의 작용들에 의하여 제품을 개념화할수 있다. 이러한 작용들은 가장 낮은 준위에 도달할 때까지 보다 더 낮은 준위의 작용들에 의하여 정의될수 있다. 이러한 준위에서 작용들은 프로그람작성언어에서 미리 정의한 구조들에 의하여 표현된다. 매 준위에서 설계자는 그 준위에 적합한 작용들에 의하여 제품을 표현하는데만 관심을 가진다. 설계자는 또한 그 아래준위를 무시할수 있다. 그 준위는 추상화의 다른 준위 즉 그다음의 세련단계에서 처리되게 될것이다. 설계자는 또한 현재의 준위를 설계하는 견지에서 관련이 없는 준위인 그우의 준위를 무시할수 있다.

7．4．2．자료교갑화와 제품의 유지정비

유지정비의 견지에서 자료교갑화문제를 취급한다면 기본문제는 앞으로 진행될 변경들의 영향을 최소화하도록 제품을 변경시키고 설계할 가능성을 줄 제품의 측면들을 파악하는것이다. 이와 같은 자료구조들은 변경되지 않을것 같다. 즉 실례로 만일 제품이 일감

대기렬들을 포함한다면 앞으로의 판본들이 그것들을 병합할것 같다. 동시에 일감대기렬들이 실현되는 특정한 방식은 잘 변화될수 있으며 자료교갑화는 그러한 변경들에 대처할수 있는 수단을 제공해 준다.

그림 7-17은 C++언어에서 **class JobQueue**와 같은 일감대기렬자료구조의 실현에 대하여 보여 주었다. 그림 7-18은 그에 대응하는 Java언어의 실현이다(다음의 《알고 싶은 문제》에는 그림 7-19부터 7-26까지는 물론 그림 7-17과 그림 7-18에 있는 프로그람작성방식에 대한 주석을 주고 있다). 그림 7-17 혹은 그림 7-18에서 대기렬은 25개의 일감번호들로 이루어 진 하나의 배렬로 실현되였다. 첫번째 요소는 queue[0]이고 25번째 요소는 queue[24]이다. 매개 일감번호는 옹근수로 표현된다. 예약어 **public**는 queuelength와 queue를 조작체계의 어디에서나 볼수 있게 해준다. 결과적인 공통결합은 아주 불충분하게 실현될것이며 7.6에서 수정될것이다.

알고 싶은 문제

이 책에서는 훌륭한 프로그람작성실천을 위하여 자료추상화문제들을 강조하는 방법으로 그림 7-17부터 그림 7-26까지의 코드실례를 신중하게 작성하였다. 실례로 그림 7-17과 7-18에서 있는 **JobQueue**의 정의에서 수자 25는 분명히 파라메터로 즉 C++언어에서는 **const**변수로 혹은 Java언어에서는 **public static final**변수로 코드작성하여야 한다. 보다 간단히 하기 위하여 자리부족(빈 대기렬에서 하나의 항목을 삭제하기 위한 시도)과 자리넘침(가득찬 대기렬에 항목을 추가하기 위한 시도)과 같은 조건들에 대한 검사들은 그쳐 지나가 버렸다. 임의의 실제적인 제품에서 이와 같은 검사들을 포함하는것은 절대적으로 필수적이다. 이밖에 언어에 고유한 특성들은 최소화하였다. 실례로 C++프로그람작성자는 queueLength의 값을 하나 증가시키기 위하여 일반적으로

```
queueLength= queueLength+1;
```

이 아니라

```
queueLength++;
```

를 리용한다. 이와 류사하게 구축자와 파괴자를 리용하는 문제도 최소화되였다.

개괄하여 말한다면 단지 교육적인 목적으로 그림 7-17부터 그림 7-26까지 코드를 작성하였다. 이 코드들은 임의의 다른 목적을 위하여 리용되지 말아야 한다.

그것들은 **public**이기때문에 **class JobQueue**에 있는 방법들은 조작체계의 그 어느 곳에서나 호출할수 있다. 특히 그림 7-19에서는 **class JobQueue**가 C++언어를 리용하여 방법 queueHandler에 의하여 어떻게 리용될수 있는가 하는것을 보여 주며 그림 7-20은 이에 대응하는 Java언어의 실현이다.

방법 queueHandler는 일감대기렬이 어떻게 실현되는가 하는 지식이 없이 **JobQueue**에 대한 방법들인 initializeJobQueue, addJobQueue, removeJobFromQueue를 호출한다. 즉 **class JobQueue**를 리용하는데 필요한 유일한 정보는 세가지 방법들과 관련한 대면부정보이다.

```
//
// 주의 :
// 이 코드는 좋은 C++ 작성방법을 리용하지 않고  C++를 전문하지 않은  독자
// 들이 리해할수 있도록  씌여 졌다. 또한 간단하게 하기 위하여 자리넘침과 자리
// 부족검사와 같은 사활적인 특징들은  빼놓았다. 자세한것은 우의 《알고 싶은
// 문제》를 보시오.
//
class  JobQueue
{
   // 속성
   public:
      int  queueLength;        // 일감대기렬의 길이
      int  queue[25];          // 대기렬은 25개까지 일감을 포함할수 있다

   // 방법
   public:
      void  initializeJobQueue ()
      /*
      * 빈 일감대기렬은 길이가 0이다
      */
      {
          queueLength  =  0;
      }

      void  addJobToQueue (int  jobNumber)
      /*
       *일감대기렬의 마감에 일감을 추가한다
       */
      {
            queue[queueLength]  =  jobNumber;
            queueLength  =  queueLength+1;
      }

      int  removeJobFromQueue ()
      /*
      *  jobNumber를 일감대기렬의 앞부분에 보관되여 있는 일감의 번호와 갈
      *  이 설정하고 일감대기렬의 앞부분에 있는 일감을 삭제한다. 그리고
      *  jobNumber를  귀환한다
      */
      {
          int  jobNumber  =  queue[0];
          queueLength  =  queueLength-1;
          for  (  int   k  =  0;  k<  queueLength;  k++)
              queue[k]  =  queue[k+1]
          return  jobNumber;
      }
}//  class   JobQueue
```

그림 7-17. **class JobQueue**의 C++실현

(**public** 속성에 의하여 제기되는 문제들은 7.6에서 해결한다.)

```
//
// 주의 :
// 이 코드는 좋은 Java 작성방법을 리용하지 않고 Java를 전문하지 않은 독자들
// 이 리해할수 있도록 씌여 졌다. 또한 간단하게 하기 위하여 자리넘침과 자리부
// 족검사와 같은 사활적인 특징들은 빼놓았다. 자세한것은 우의 《알고 싶은 문
// 제》를 보시오.
//
class JobQueue
{
   // 속성
   public int queueLength;               // 일감대기렬의 길이
   public int queue[] = new int[25];     // 대기렬은 25개까지 일감을
                                         // 포함할수 있다

   // 방법
     public void initializeJobQueue ()
     /*
      * 빈 일감대기렬은 길이가 0이다
      */
     {
         queueLength = 0;
     }

     public void addJobToQueue (int jobNumber)
     /*
      * 일감대기렬의 마감에 일감을 추가한다
      */
     {
       queue[queueLength] = jobNumber;
       queueLength = queueLength+1;
     }

     public int removeJobFromQueue ()
     /*
      *  jobNumber를 일감대기렬의 앞부분에 보관되여 있는 일감의 번호와
      *  같이 설정하고 일감대기렬의 앞부분에 있는 일감을 삭제한다. 그리고
      *  jobNumber를 귀환한다
      */
     {
        int jobNumber = queue[0];
        queueLength = queueLength-1;
        for ( int  k = 0; k< queueLength; k++)
            queue[k] = queue[k+1];
        return jobNumber;
     }
}// class JobQueue
```

그림 7-18. **class JobQueue**의 Java실현

(**public** 속성에 의하여 제기되는 문제들은 7.6에서 해결한다.)

```
class Scheduler
{
      …
      public:
        void queueHandler ()
         {
            int   jobA, jobB;
            JobQueue     jobQueueJ;
              //   명령문들
            jobQueueJ.initializeJobQueue ();
            jobQueueJ.addJobToQueue (jobA);
             jobB  =  jobQueueJ.removeJobFromQueue ();
              // 그밖의   명령문들
           }// queueHandler
       …
}// class Scheduler
```

그림 7-19. queueHandler의 C++실현

```
class Scheduler
{
      …
      public void queueHandler ()
      {
         int jobA, jobB;
         JobQueue jobQueueJ = new  JobQueue ();
           //  명령문들
          jobQueueJ.initializeJobQueue ();
          jobQueueJ.addJobToQueue (jobA);
          jobB  =  jobQueueJ.removeJobFromQueue ();
           // 그밖의   명령문들
      }// queueHandler
           …
}// class Scheduler
```

그림 7-20. queueHandler의 Java실현

이제 일감대기렬이 현재는 일감번호들의 선형목록으로서 실현되였지만 그것을 쌍방향으로 련결된 일감기록들의 목록으로서 재실현하도록 결정되였다고 가정하자. 매개 일감기록은 세개의 구성요소를 가질것이다. 즉 그 구성요소들은 앞에서와 마찬가지로 일감번호와 련결목록의 앞에 있는 일감기록에 대한 지적자와 그뒤에 있는 일감기록에 대한 지적자들이다. 이것은 그림 7-21와 그림 7-22에서 보여 주는것처럼 각각 C++와 Java로 서술되였다. 이제 일감대기렬이 실현되는 방식에 대한 이러한 수정의 결과로서 전반적인

쏘프트웨어제품에 대하여서는 어떤 변경을 진행해야 하는가? 사실상 **JobQueue**만이 변경되여야 한다. 그림 7-23에서는 그림 7-21의 쌍방향으로 련결된 목록을 리용하여 **JobQueue**에 대한 C++언어실현의 륜곽을 보여 주었다. **JobQueue**와 제품의 나머지(방법 **queueHandler**를 포함하여) 부분들사이의 대면부가 그 어떤 방식으로도 변경되지 않았다는 사실을 강조하기 위하여 실현세부들을 공개하지 않았다(문제 7.10을 보라). 즉 세가지 방법들인 **initalizeJobQueue, addJobToQueue, removeJobFromQueue**은 정확히 이전과 같은 방법으로 호출된다. 특히 방법 **addJobFromQueue**를 불러 낼 때 그것은 여전히 **JobQueue**에 옹근수값을 넘겨 주며 **removeJobFromQueue**는 비록 일감대기렬 그자체가 전혀 다른 방식으로 실현되였다고 하더라도 **JobQueue**로부터 옹근수값을 귀환한다. 결국 방법 **queueHandler**의 원천코드는 전혀 변경시킬 필요가 없다. 이리하여 자료교갑화는 제품의 유지정비를 간단하게 하고 회귀오유가 발생할 기회를 줄이는 방식으로 자료추상화의 실현을 지원하고 있다.

```
class  JobRecord
{
   public:
        int             jobNo;      // 일감번호(옹근수)
        JobRecord  *inFront;  // 맨 앞쪽의 일감기록에 대한 지적자
        JobRecord  *inRear;   // 맨 뒤쪽의 일감기록에 대한 지적자
}//  class  JobRecord
```

그림 7-21. 쌍방향으로 련결된 **class JobRecord**의 C++명세서
(**public** 속성에 의하여 제기되는 문제들은 7.6에서 해결한다.)

```
class  JobRecord
{
   public  int             jobNo;      // 일감의 번호(옹근수)
   public  JobRecord  inFront;    // 맨 앞쪽의 일감기록에 대한 지적자
   public  JobRecord  inRear;     // 맨 뒤쪽의 일감기록에 대한 지적자
}//  class  JobRecord
```

그림 7-22. 두 방식으로 련결된 class JobRecord의 Java명세서
(**public** 속성에 의하여 제기되는 문제들은 7.6에서 해결한다.)

그림 7-17과 7-18 그리고 그림 7-19와 7-20을 비교하여 보면 이 실례들에 대한 C++와 Java실현들사이의 차이점은 본질에 있어서 문장론적이라는것이 명백하다. 이 장의 나머지 부분들에서 우리는 기타 다른 실현에서의 문장론적차이들에 대한 설명과 함께 하나의 실현만을 제시하였다. 특히 일감대기렬코드의 나머지는 C++로 작성되였고 기타 다른 모든 코드실례들은 Java로 작성되였다.

```
class JobQueue
{
    public:
      JobRecord *frontOfQueue;    // 대기렬의 맨 앞쪽에 대한 지적자
      JobRecord *rearOfQueue;     // 대기렬의 맨 뒤쪽에 대한 지적자

      void initializeJobQueue ()
      {
        /*
         * frontOfQueue와 rearOfQueue를 NULL로 설정하여 일감대기렬을
         * 초기화한다
         */
      }

      void addJobToQueue (int jobNumber)
      {
      /*
       * 새 일감기록을 창조하고 jobNo마당에 jobNumber를 넣어 준다.
       * 현재의 rearOfQueue를 지적하기 위하여(대기렬의 마감에 새 일감을
       * 련결하여) inFront항목을 설정한다.
       * inRear항목을  NULL로 설정한다
       * 새 기록을 지적하기 위하여(쌍방향으로 련결을 설정하여)
       * 지적된 기록의 inRear항목을 현재의 rearOfQueue로 설정한다
       * 마지막으로 이 새 기록을 지적하기 위하여 rearOfQueue 를 설정한다.
       */
      }

      int removeJobFromQueue ()
      {
        /* jobNumber을 대기렬의 맨앞에 있는 기록의  jobNo항목과 같게 설
         * 정한다. 대기렬의 다음 항목을 지적하기 위하여 frontOfQueue를 갱신
         * 한다. 대기렬의 지금의 앞에 있는 기록의  inFront항목을  NULL로
         * 설정한다. 그리고  jobNumber 를 귀환한다
         */
      }
}// class JobQueue
```

그림 7-23. 쌍방향으로 련결된 목록을 리용한 **class JobQueue**의 C++ 일반적실현

7. 5. 추상자료형

그림 7-17(혹은 그림 7-18)은 일감대기렬 **class** 즉 자료형과 그러한 자료형에 대하여 수행하게 되는 작용들의 실현이다. 그러한 구조를 추상자료형(*abstract data type*)이라고 부른다.

그림 7-24는 이 추상자료형이 조작체계의 세개의 일감대기렬에 대하여 C++로 어떻게 리용할수 있는가를 보여 준다. 세개의 일감대기렬들은 실체화된다. 즉 highPriorityQueue,

mediumPriorityQueue, lowPriorityQueue로 실체화된다(Java판본에서는 세개의 일감대기렬들에 대한 자료선언에서 문장론적인 차이가 있다.). 명령문 highPriorityQueue, initializeJobQueue는 《조작 initializeJobQueue를 자료구조 highPriorityQueue에 적용하는것》을 의미하며 다른 두개의 명령문들에 대하여도 이와 류사하게 고찰할수 있다.

```
class Scheduler
{
  …
   public:
      void queueHandler ()
      {
           int          job1, job2;
          JobQueue  highPriorityQueue;
          JobQueue  mediumPriorityQueue;
          JobQueue  lowPriorityQueue;
           // 명령문들
           highPriorityQueue.initializeJobQueue ();
           mediumPriorityQueue.addJobToQueue (job1);
           job2 = lowPriorityQueue.removeJobFromQueue ();
           // 그밖의 명령문들
       }// queueHandler
           …
}// class Scheduler
```

그림 7-24. 그림 7-17의 추상자료형을 리용하여 실현한 C++ 방법 queueHandler

```
class Rational
{
    …
      public int numerator;
      public int denominator;
      public void sameDenominator (Rational r, Rational s)
      {
        // r와  s를  같은 분모로 통분하는  코드
      }
      public boolean equal (Rational t, Rational u)
      {
        Rational v, w;
        v=t;
        w=u;
        sameDenominator (v,w)
        return (v.numerator == w.numerator);
      }
      //  두 유리수를 더하기, 덜기, 곱하기, 나누기방법
}// class Rational
```

그림 7-25. 유리수의 Java추상자료형실현
(**public** 속성에 의하여 제기되는 문제들은 7.6에서 해결한다.)

추상자료형은 널리 적용할수 있는 설계도구이다. 실례로 수많은 작용들이 유리수들 즉 n/d형식으로 표현될수 있는 수들에 대하여 수행되여야 하는 제품을 개발한다고 하자. 여기서 n과 d는 옹근수들이고 d ≠ 0이다. 유리수들은 하나의 1차원옹근수배렬의 두개의 요소 또는 한 클라스의 두개의 속성과 같은 다양한 방식으로 표현될수 있다. 추상자료형으로 유리수들을 실현하기 위하여 자료구조에 대한 적합한 표현이 선택된다. Java언어에서 그것은 두개의 옹근수들로부터 유리수를 구성하고 두개의 유리수들을 더하거나 곱하는것과 같은 유리수들에 대한 여러가지 작용들과 함께 그림 7-25에서와 같이 정의할수 있을것이다(그림 7-25에서 numerator와 denominator와 같은 **public**속성에 의하여 발생되는 문제들은 7.6에서 수정될것이다.). 대응하는 C++언어에서의 실현은 예약어 **public**의 배치에서 차이난다. 또한 인수가 참조로 넘겨 질 때 &가 요구된다.

추상자료형은 자료추상화와 처리절차추상화를 모두 지원한다(7.4.1). 이밖에 제품이 수정될 때 추상자료형은 변화되지 않는다. 최악의 경우에 추가적인 작용들은 추상자료형에 추가되여야 한다. 따라서 제품개발과 제품의 유지정비의 견지에서 볼 때 추상자료형들은 쏘프트웨어개발자들에게는 매력 있는 도구로 된다.

7. 6. 정보은페

7.4.1에서 론의된 두가지 형태의 추상화(자료추상화와 처리절차추상화)들은 파나스(Parnas)가 제기한 보다 일반적인 설계개념 즉 정보은페(*information hiding*)의 실체들이다 [Parnas, 1971, 1972a, 1972b]. 파나스의 착상은 장래의 유지정비를 지향하고 있다. 제품이 설계되기전에 앞으로 변경될 가능성이 있는 실현들에 대한 목록이 작성되여야 한다. 그때 모듈들은 결과적인 설계의 실현세부들이 다른 모듈들에게 숨겨 지도록 설계되여야 한다. 그러므로 장래에 진행될 매 변경들은 하나의 특정한 모듈에 국한되게 된다. 원래의 실현결정에 대한 세부들은 다른 모듈들에서는 볼수 없게 되므로 설계를 변경시키는것은 그 어떤 다른 모듈들에는 명백히 영향을 주지 않는다(정보은페에 대하여 좀 더 자세히 알자면 다음에 서술되여 있는 《알고 싶은 문제》를 보시오).

알고 싶은 문제

용어 **정보은페**(*information hiding*)는 무엇인가 잘못된 이름인것 같다. 왜냐하면 은페하여야 할것은 정보가 아니라 실현세부이기때문에 보다 정확하게 서술하면 세부은페 *details hiding*)라고 해야 할것 같다.

이러한 착상이 실천적으로 어떻게 리용될수 있는가를 보기 위하여 그림 7-17에서의 추상자료형실현을 리용한 그림 7-24를 고찰하자. 추상자료형을 리용하는 기본리유는 일감대기렬의 내용들이 그림 7-17의 세 조작들중 어느 하나를 불러냄으로써만 변경될수 있게 하는것이다. 유감스럽게도 그 실현의 본질은 일감대기렬들이 다른 방식들로 변경될수 있게 하는것과 같다.

속성들인 queueLength와 queue는 둘다 그림 7-17에서 **public**로 선언되여 있고 따라서 queueHandler안에서 접근가능하다. 결과 그림 7-24에서 highPriorityQueue를 변경시키기 위하여 queueHandler안의 그 어디에서나

highPriorityQueue.queue[7] = －5678;

과 같은 대입명령문을 리용하는것은 C++(혹은 Java) 에서 완전히 합법적이다. 달리 말하여 일감대기렬의 내용들은 추상자료형의 세 조작들중 아무것도 리용하지 않고 변경시킬수 있다. 응집도를 낮추고 결합도를 강하게 하는것과 관련하여 가질수 있는 의미외에도 관리자는 제품이 7.3.2에서 설명한것처럼 콤퓨터범죄를 범할수 있다는것을 인식하여야 한다.

```
class JobQueue
{
   // 속성
   public int queueLength;                 // 일감대기렬의 길이
   public int queue[] = new int[25];       // 대기렬은 25개까지 일감을 포함
                                            // 할수 있다.

   // 방법
   public void initializeJobQueue ()
   {
     //  그림 7-17에서 변화되지 않은 방법의 체부
   }

   public void addJobToQueue (int jobNumber)
   {
     //  그림 7-17에서 변화되지 않은 방법의  체부
   }

   public int removeJobFromQueue ()
   {
     //  그림 7-17에서 변화되지 않은 방법의  체부
   }
}// class JobQueue
```

그림 7-26. 그림 7-17, 7-18, 7-21, 7-22, 7-25의 문제들을 정확히 하는 정보은폐를 진행한 C++ 추상자료형실현

다행히 한가지 출로가 있다. C++와 Java의 설계자들은 클라스명세서안에서 정보은폐를 규정하였다. 이것은 그림 7-26에서 C++에 관하여 보여 주었다(Java와의 문장론적차이는 이전과 같다.).

public에서 **private**로 속성들의 가시성수식어를 변경시킨것외에는 그림 7-26은 그림 7-17과 같다. 지금 다른 모듈들에서 볼수 있는 유일한 정보는 **JobQueue**가 하나의 클라스류형이며 명기된 대면부를 가진 세 동작들은 결과적인 일감대기렬에 작용할수 있다는

것이다. 그러나 일감대기렬들이 실현되는 정확한 방도는 **private**화되였다. 즉 외부에서 볼 수 없다. 그림 7-27에 있는 도표는 **private**속성을 가진 클라스가 C++나 Java사용자가 완전한 정보은페를 가진 추상자료형을 어떻게 실현할수 있는가 하는것을 보여 준다.

정보은페기법은 또한 7.3.2의 마감에서 언급한것처럼 공통결합을 방지하는데 리용될 수 있다. 여기서 설명된 제품 즉 55개의 서술자들에 의하여 명기된 페트롤렌저장탕크들을 위한 콤퓨터지원설계도구에 대하여 다시 생각해 보자. 만일 제품이 서술자를 초기화하기 위한 **private**작용들을 가지고 실현된다면 그 어떤 공통결합도 없다. 이 해결형식은 다음절에서 설명되는것처럼 객체들이 정보은페를 지원하기때문에 객체지향파라다임으로 특징지어 진다. 이것은 객체기법을 리용하는 또 하나의 우월성으로 된다.

일감처리계획작성프로그람

```
{
    int                      job1, job2;
         :                        :
    highPriorityQueue.initializeJobQueue ();
         :                        :
    mediumPriorityQueue.addJobToQueue (job1);
         :                        :
    job2 = lowPriorityQueue.removeJobFromQueue ();
         :                        :
}
```

JobQueue

queue
queueLength
initializeJobQueue
addJobToQueue
removeJobFromQueue
의 실현세부

대면부정보관리
initializeJobQueue
addJobToQueue
removeJobFromQueue

JobQueue밖에서는 보이지 않는다 **JobQueue밖에서 보인다**

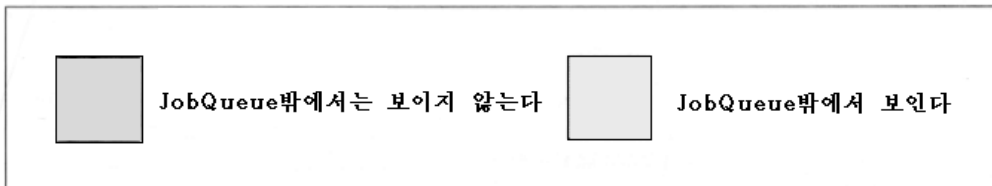
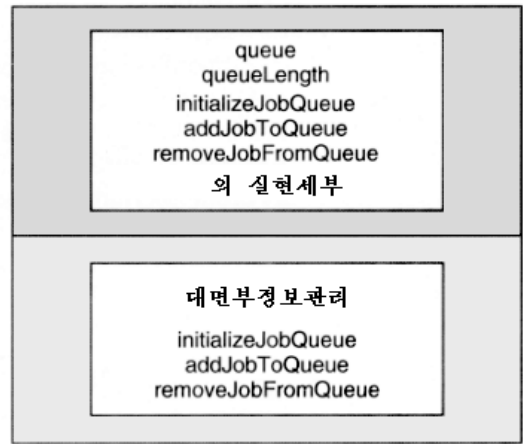



그림 7-27. **private**속성으로 정보은페를 진행한 추상자료형의 표현
(그림 7-24와 함께 그림 7-26)

7. 7. 객체

이 장의 앞부분에서 언급된것처럼 객체들은 단순히 그림 7-28에 보여 준 과정에서 다음의 단계로 된다. 객체들에 대하여서는 그 어떤 특별한것이 없다. 즉 그것들은 추상자료형이나 정보적인 응집도를 가진 모듈들만큼 일반적인것이다. 객체의 중요성은 그것들이 자기자신의 추가적인 속성들은 물론 그림 7-28에 있는 자기 선조들의 모든 속성들을 가지고 있다는것이다.

높은 응집력과 낮은 결합력을 가진 객체 (7.9절)
⇑
객체 (7.7절)
⇑
정보은폐 (7.6절)
⇑
추상자료형 (7.5절)
⇑
자료교갑화 (7.4절)
⇑
높은 응집력과 낮은 결합력을 가진 모듈 (7.2절,7.3절)
⇑
모듈 (7.1절)

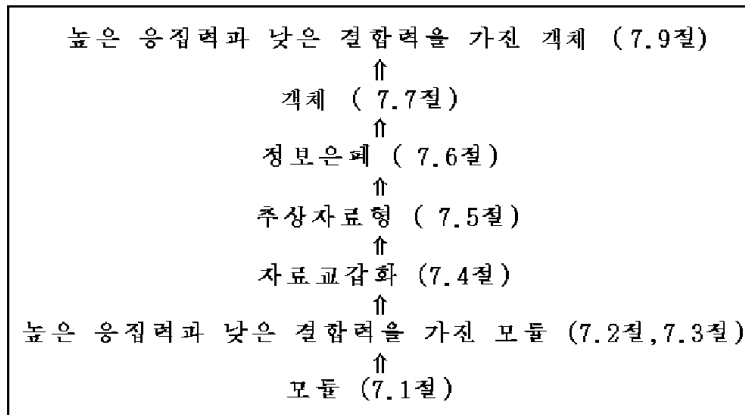



그림 7-28. 7장의 기본개념

객체에 대한 불완전한 정의는 객체가 추상자료형의 실체이라는것이다. 즉 제품은 추상자료형들에 의하여 설계되며 제품의 변수들(객체들)은 추상자료형들의 실체이다. 그러나 객체를 추상자료형의 실체라고 정의한것은 너무 단순하다. 이와 관련하여 무엇인가 더 필요하다. 즉 Simula 67 [Dahl and Nygaard, 1966]에서 처음으로 소개된 개념인 계승(*inheritance*)이 필요하게 된다. 계승은 Smalltalk [Goldberg and Robson, 1989], C++ [Stroustrup, 1991], Eiffel [Meyer, 1992b], Ada95 [ISO/IEC 8652, 1995], Java [Flanagan and Loukides, 1997]와 같은 모든 객체지향프로그람작성언어들에 의하여 지원된다. 계승의 리면에 있는 기본착상은 새로운 자료형들이 령상태에서부터 정의되는것이 아니라 이전에 정의된 형들의 확장으로 정의될수 있다는것이다[Meyer, 1986].

객체지향언어에서 클라스(*class*)는 계승을 지원하는 추상자료형으로 정의할수 있다. 그때의 객체는 클라스의 실체로 된다. 클라스들이 어떻게 리용되는가를 알기 위하여 다음의 실례를 고찰하자. **HumanBeing**을 클라스로, Joe를 그 클라스의 실체인 객체라고 정의하자. 매 **HumanBeing**은 나이와 키와 같은 일정한 속성을 가지고 있으며 값들은 객체 Joe에 대하여 설명할 때 그 속성들에 대입할수 있다. 이제 **Parent**를 **HumanBeing**의 부분클라스(*subclass*)(도출클라스 또는 파생클라스; *derived class*)로 정의한다고 가정하자. 이것은 **Parent**가 **HumanBeing**의 모든 속성들을 가지고 있으며 그밖에 제일 나이 많은 자식(맏자식)의 이름과 아이들의 수와 같은 자기 고유의 속성들을 가질수 있다는것을 의미한다. 이에 대해서는 그림 7-29에서 설명하고 있다. 객체지향용어에서는 **Parent** *isA* **HumanBeing**이라고 한다. 이리하여 그림 7-29에서 화살표는 방향이 잘못된것 같다. 사실상 화살표는 *isA*관계를 나타내고 있고 따라서 파생클라스로부터 기초클라스에로 가리킨다(계승을 표시하기 위하여 열린 화살을 리용한것은 UML규정이다. 그리고 또 다른것은 클라스명들이 매개 단어의 첫 글자를 대문자로, 굵은체로 나타낸다는것이다. UML은 12장에서 좀 더 자세히 론의한다.).

클라스 **Parent**는 기초클라스 **HumanBeing**의 파생클라스(혹은 부분클라스)이기때문에 **HumanBeing**의 모든 속성들을 계승한다. 만일 Fred가 객체이고 클라스 **Parent**의 실체이라면 Fred는 **Parent**의 모든 속성들을 가지고 있으며 또한 **HumanBeing**의 모든 속성들을

계승한다.

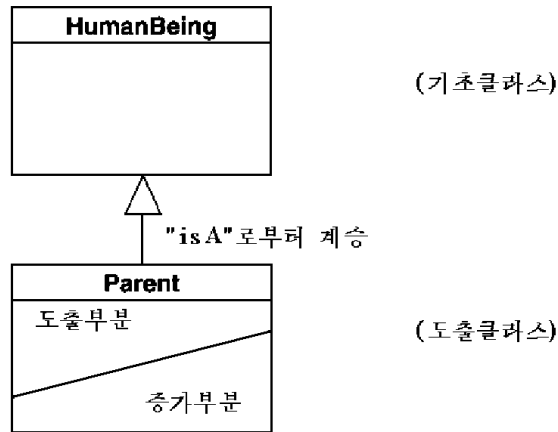



그림 7-29. 도출된 형들과 계승

그림 7-30에 Java의 실현을 보여 주었다. C++판본은 **Private**와 **Public**수식어들의 배치가 다르다. 또한 이 실례에서 Java문법 **extends**은 C++에서 **: public**로 교체된다.

```
class HumanBeing
{
  private int    age;
  private froat height;
  // HumanBeing 에 대한 조작의 public선언
}// class HumanBeing

class Parent extends HumanBeing
{
  private string nameOfOldestChild;
  private int     numberOfChildren;
  // Parent대한 조작의 public선언
}// class Parent
```

그림 7-30. 그림 7-29의 Java실현

계승의 성질은 모든 객체지향프로그람작성언어들의 본질적인 특성이다. 그러나 계승이나 클라스의 개념은 C, COBOL 혹은 FORTRAN과 같은 고전적인 언어들에서는 지원되지 않는다. 따라서 객체지향파라다임은 이러한 대중적인 언어들에 의하여 직접 실현될수 없다.

객체지향파라다임의 용어에는 그림 7-29에 있는 **Parent**와 **HumanBeing**사이의 관계를 고찰하기 위한 두가지 서로 다른 방법들이 있다. **Parent**는 **HumanBeing**의 특수화라고 말

할수 있으며 혹은 **HumanBeing**이 **Parent**의 일반화라고 말할수 있다. 특수화와 일반화외에도 클라스는 두가지 다른 기본관계 즉 집합과 결합을 가진다[Blaha, Premerlani, and Rumbaugh, 1988]. 집합(*aggregation*)은 클라스의 구성요소를 의미한다. 실례로 클라스 **PersonalComputer**는 구성요소들 즉 CPU, Monitor, Keyboard, Printer 등으로 구성되여 있다. 이것을 그림 7-31에 보여 주었다(집합을 보여 주기 위하여 다이아몬드글자체(보통 활자들중에서 가장 작은 활자들중의 하나)를 리용하는것은 또 하나의 다른 UML규정이다). 이에 대해서는 새로운것이 전혀 없다. 즉 어떤 언어가 C에서 **struct**와 같은 기록들을 지원할 때마다 이러한 현상이 발생한다. 그러나 객체지향의 범위내에서 관련항목들을 묶는데 리용되며 결과적으로 재리용가능한 클라스를 생성한다(8.1).

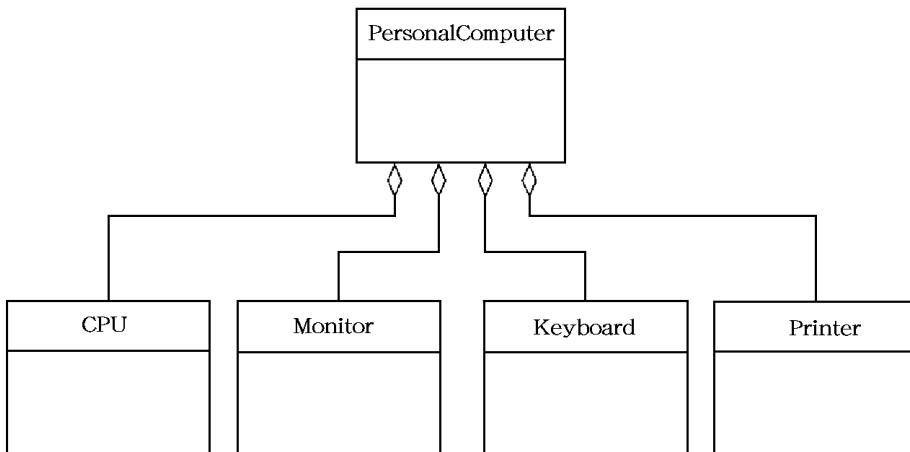



그림 7-31. 집합의 실례

결합(*association*)은 두개의 명백한 련관이 없는 클라스들사이의 일종의 관계를 의미한다. 실례로 렌트겐의사와 미술가사이에는 그 어떤 련관이 없는것 같다. 그러나 MRI기계가 어떻게 동작하는가를 설명한 책의 도식들을 그리는것과 관련하여 렌트겐의사는 미술가에게 의견을 물어 볼수 있다. 결합은 그림 7-32에 보여 주고 있다.

이 실례에서 결합의 본질은 단어 상담(*consults*)에 의하여 지적된다. 이밖에 검은 삼각형은 결합의 방향을 가리킨다. 결국 발목이 부러진 미술가는 렌트겐의사와 상담할수 있다.

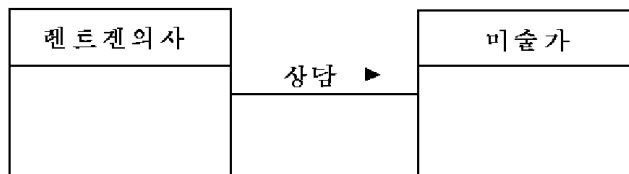



그림 7-32. 결합의 실례

다른 객체지향언어들에서와 마찬가지로 Java와 C++표기법에서의 한가지 측면은 작용과 자료의 등가성을 명백히 반영한다. 우선 기록을 지원하는 고전적인 언어 실례로 C언어를 고찰하자. record_1은 **struct**(기록)이고 field_2는 클라스내의 마당이라고 가정하자. 그러면 마당은 record_1.field_2로 간주된다. 즉 .은 기록내에서 성원관계를 나타낸다. 만일 function_3이 C모듈내에서의 하나의 함수이라면 function_3()은 그 함수의 호출을 의미한다.

대조적으로 **ClassA**가 속성 **attributeB**와 방법 **methodC**를 가진 하나의 클라스라고 가정하자. 더 나아가서 ourObjcct가 **ClassA**의 실체라고 가정하자. 그러면 마당은 ourObject.attributeB로 간주된다. 그리고 ourObject.methodC()는 이 방법의 호출을 나타낸다. 그리하여 .은 성원요소가 속성이든 방법이든 객체내에서의 성원관계를 나타내는데 리용된다.

객체들(혹은 클라스들)을 리용하는 우월성은 정확히 자료추상화와 처리절차추상화를 비롯한 추상자료형을 리용하는데서의 우월성이다. 이밖에 클라스들의 계승측면은 자료추상화의 그이상의 계층을 제공하며 보다 쉽고 보다 적은 오유를 가지는 제품개발을 할수 있도록 한다. 또 하나의 우월성은 다음절의 주제로 되는 다형성과 동적맺기를 계승과 결합하는데로부터 나온다.

7. 8. 계승, 다형성과 동적맺기

어떤 콤퓨터의 조작체계가 파일을 열기 위하여 호출된다고 하자. 파일은 여러가지 각이한 매체들에 저장될수 있다. 실례로 파일은 디스크파일, 테프파일 혹은 유연성자기디스크파일일수 있다. 구조화파라다임을 리용하면 세개의 함수들 즉 open_disk_file, open_tape_file, open_diskette_file이 있을수 있다. 이에 대하여서는 그림 7-33의 ㄱ)에 보여 주었다. 만일 my_file이 파일로 선언되면 그것이 디스크파일인지, 테프파일인지 혹은 플로피자기디스크파일인지를 시험해 보는것이 중요하다.

반대로 객체지향파라다임이 리용될 때 세개의 파생클라스들인 **DiskFileClass, TapeFileClass, DisketteFileClass**를 가진 **FileClass**라는 클라스가 정의된다. 이에 대하여 그림 7-33의 ㄴ)에 보여 주었다. 여기서 열린 화살표가 계승을 나타낸다는것을 상기하시오.

이제 방법 open이 어미클라스 **FileClass**로 정의되고 세개의 파생클라스들에 의하여 계승된다고 하자. 유감스럽게도 세개의 각이한 형식의 파일들을 여는데 서로 다른 작용들이 수행되여야 하기때문에 이것은 동작할수 없다.

그 해결책은 다음과 같다. 어미클라스 **FileClass**에서 가상적인 방법 open을 선언한다. Java에서는 이러한 방법을 **abstract**로 선언한다. C++에서는 대신 예약어 **virtual**을 리용한다. 이 방법의 하나의 특정한 실현은 세개의 파생클라스에서 각각 출현하며 매 방법은 그림 7-33의 ㄴ)에서 보여 준것처럼 open이라는 동일한 이름을 가진다. 이제 myFile이 파일로 선언된다고 하자. 실행시에 통보문 myFile.open()이 보내여 진다. 이제 객체지향체계는 myFile이 디스크파일인지, 테프파일인지 혹은 플로피자기디스크파일인지를 결정하고 open에 대한 적절한 판본을 호출한다.

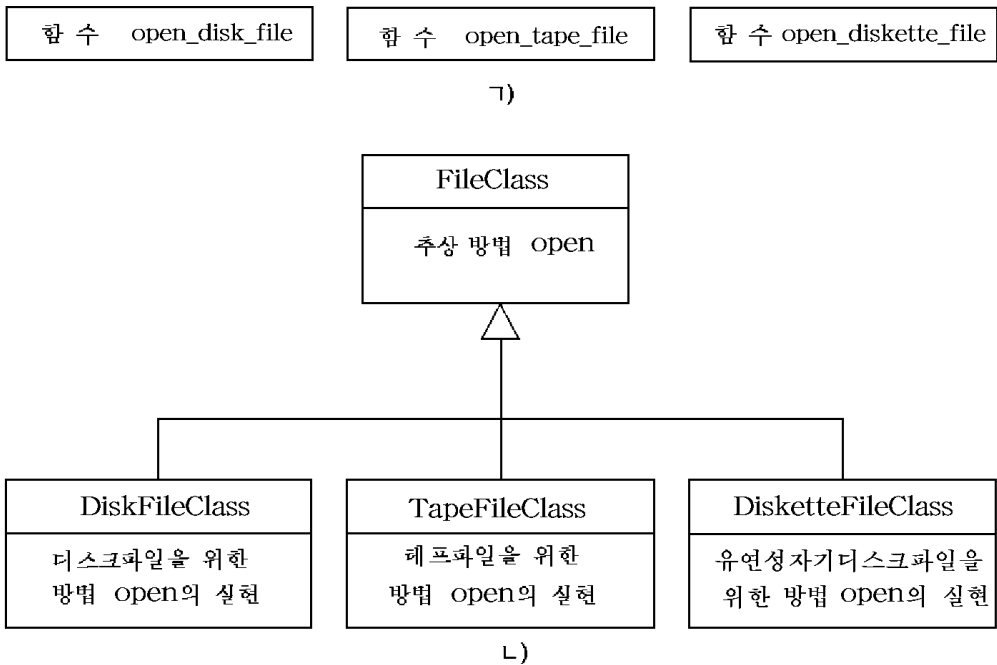



그림 7-33. 파일열기에 요구되는 작용
ㄱ) 구조화실현, ㄴ) Java표기법을 리용한 객체지향파일클라스의 계층

즉 체계는 실행시에 객체 myFile이 클라스 **DiskFileClass**, 클라스 **TapeFileClass** 혹은 클라스 **DisketteFileClass**의 실체인가를 결정하고 자동적으로 정확한 방법을 호출한다. 이것은 실행시에(동적으로) 진행되여야 하며 콤파일시에는(정적으로) 진행되지 말아야 하기때문에 객체를 적당한 방법에 련결시키는 작용을 동적맺기(*dynamic binding*)라고 한다. 더우기 방법 open이 각이한 클라스들의 객체들에 적용될수 있기때문에 그것을 다형성(*polymorphic*)이라고 한다. 이 용어는 《많은 형태》를 의미한다. 탄소결정편들이 굳은 금강석들과 무른 흑연을 비롯한 여러가지 각이한 형태들로 나타나는것처럼 방법 open도 세개의 각이한 판본으로 나타난다. Java에서 이 판본들은 DiskFileClass.open, TapeFileClass.open 그리고 DisketteFileClass.open으로 표시된다(C++에서 .은 두개의 :으로 교체되며 파일들은 DiskFileClass::open, TapeFileClass::open, DisketteFileClass::open으로 표시된다). 그러나 동적맺기로 인하여 특정한 파일을 열기 위하여 어느 방법을 호출할것인가 하는것을 결정하는것은 필요 없다. 대신 실행시에 오직 통보문 myFile.open()만을 보내는것이 필요하며 체계는 myFile의 형(클라스)을 결정하고 정확한 방법을 호출할것이다.

이러한 착상은 바로 **abstract**(**virtual**)방법에서 더 유용하다. 그림 7-34에 보여 준바와 같이 클라스의 계승을 고찰하자. 모든 클라스들은 기초클라스로부터 계승에 의해서 파생된다. 방법 checkOrder(b:**Base**)가 클라스 **Base**의 하나의 실체를 인수로서 취한다고 가정하자. 그러면 계승, 다형성, 동적맺기의 결과로 checkOrder를 클라스 **Base**뿐만아니라 그로

부터 파생된 임의의 클라스의 인수와 함께 호출하는것은 타당하다.

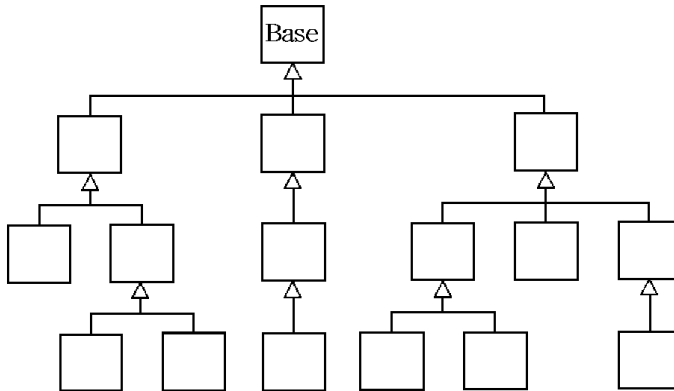



그림 7-34. 클라스의 계층

여기서 필요한것은 checkOrder를 호출하는것이며 실행시에 모든것에 주의를 돌리는 것이다. 이러한 기법은 쏘프트웨어전문가들이 어떤 통보문이 보내여 질 때 인수의 정확한 형에는 관심할 필요가 없다는 점에서 아주 강력한 기법으로 된다.

그러나 다형성과 동적맺기는 중요한 결함으로도 된다. 첫째로, 실행중에 특정한 다형성방법들가운데서 어느 판본을 호출하겠는가를 콤파일할 때 결정하는것은 일반적으로 불가능하다. 따라서 실패의 원인을 결정하기가 대단히 어려워 진다.

둘째로, 다형성과 동적맺기는 유지정비에 부정적인 영향을 미친다. 유지정비프로그람작성자의 첫째가는 임무는 일반적으로 제품을 리해하는것이다(16장에서 설명한바와 같이 유지정비자는 때로 코드를 작성하는 사람으로 되기도 한다.). 그러나 특정한 방법에 대하여 여러개의 가능성이 존재하면 곤난하게 될수 있다. 프로그람작성자는 특정한 코드부분을 동적으로 호출하는 방법들에 대해서 가능한껏 모두 고찰해야 하는데 이것은 시간이 소비되는 과제이다. 이처럼 다형성과 동적맺기는 객체지향파라다임에 우단점을 부가한다.

1.6에서 제기한 객체지향파라다임의 우월성에 대한 리유는 개념적 및 물리적독립성을 포함하고 있다. 이러한 독립성을 측정하기 위하여 객체의 범위내에서 응집도와 결합도의 개념을 다시 검토하여야 한다.

7. 9. 객체의 응집도와 결합도

객체는 일종의 모듈이다. 결국 응집도가 강하고 결합도가 약한 모듈과 관련하여 7.2과 7.3에서 제기한 문제들은 객체들에도 마찬가지로 적용된다. 이때 객체지향파라다임안에서 특정한 형태의 응집과 결합이 발생하겠는가 하는 문제가 제기된다.

우선 응집도를 고찰하자(7.2에서 설명한바와 같이 모듈의 응집도는 그 모듈에 의하여 수행되는 작용들이 기능적으로 관계되는 정도이다는것을 상기하시오.). 클라스는 두가지 작용을 포함할수 있는데 하나는 계승된 방법이고 다른 하나는 그 클라스에 고유한 방

법이다. 클라스의 응집도는 그러한 기능의 원천과는 관계없이 그 기능자체로부터 결정된다. 이것을 고찰하기 위하여 두 클라스가 같은 기능을 수행한다고 가정하자. 그러나 하나의 클라스는 자기의 상위클라스(계층도에서 그의 선조)로부터 모든 기능을 계승한다. 한편 다른 클라스들은 아무것도 계승하지 않는다. 두 클라스가 다 같은 기능을 수행하기때문에 그것들은 같은 준위의 응집도를 가진다. 달리 말하면 그 어떤 류형의 응집도도 클라스에 특정한것이 아니다. 반면에 응집도는 클라스를 비롯한 모든 형태의 모듈들에 대하여 동등하게 적용된다[Schach, 1996].

결합과 관련하여서는 우선 계승이 무시된다. 계승이 없는것으로 하여 두 클라스사이의 결합도는 고전적인 파라다임에서와 같이 결정될수 있다. 실례로 클라스의 속성이 **public**로 정의되면(제품안에서 모든 다른 부분들을 호출할수 있는) 이것은 공통적인 결합을 초래할수 있다. 놀라운것은 계승이 새로운 형태의 결합을 초래하지 않는다는것이다. 즉 결합의 결과로 발생하는 결합의 형태들은 고전적인 파라다임들에서도 역시 발생한다고 볼수 있다[Binkley and Schach, 1997]. 그러므로 고전적인 파라다임과 객체지향파라다임사이에 중요한 차이가 있음에로 불구하고 객체지향파라다임은 새로운 형의 응집도이라든가 결합도를 생성할수 없다.

그러나 객체지향파라다임에 고유한 여러가지 척도들이 제시되였다. 실례로 계승나무의 높이를 들수 있다[Chidamber and Kemerer, 1994]. 이러한 일부 척도들은 리론적이고 경험적인 토대우에서 문제시되였다[Binkley and Schach, 1996, 1997]. 이 경우에는 여전히 객체지향파라다임쏘프트웨어에 동등하게 응용될수 있는 고전적척도(응집도와 결합도와 같은)가 아니라 특별히 객체지향척도가 필요하다는것을 보여 준다.

객체지향파라다임에 대한 론의로 이 장을 결속한다.

7．１０．객체지향파라다임

매개 쏘프트웨어제품을 두가지 방법으로 고찰할수 있다. 한가지 방법은 국부변수나 대역변수, 인수, 동적자료구조, 파일 등을 비롯하여 바로 자료들을 고찰하는것이다. 다른 한가지 방법은 처리절차나 함수와 같은 자료를 처리하는 작용을 고찰하는것이다. 쏘프트웨어를 자료와 작용으로 나누어 고찰함으로써 구조화기법은 본질적으로 두 그룹으로 갈라 진다. 작용지향기법은 우선 제품의 작용들을 고찰한다. 자료는 2차적인 문제로 된다. 자료는 제품에 대한 작용을 심도 있게 해석한 다음에만 고찰된다. 반대로 자료지향기술은 제품의 자료를 중요시한다. 한편 작용들은 자료의 틀거리안에서만 고찰된다.

자료지향이나 작용지향방법의 기본 약점은 자료나 작용이 같은 자격을 가지는 두가지 측면을 가진다는것이다. 다시말하여 작용이 실행되지 않으면 자료는 변경될수 없고 자료와 결합되지 않은 작용에 대하여서도 생각할수 없다는것이다. 따라서 자료와 작용에 대하여 똑같이 중시하고 고찰해야 한다. 객체지향기법은 이러한 고찰에 기초하고 있다. 결국 객체는 자료와 작용을 모두 포함하고 있다. 객체가 추상자료형(더 정확하게 클라스)의 한가지 실례이라는것은 이미 알고 있다. 그러므로 자료와 그러한 자료를 처리하는 작용은 하나로 병합시켜 보아야 한다. 객체에서 자료는 속성이나 상태변수, 실체변수, 마당

또는 자료성원들이다. 작용은 방법(*method*) 또는 방법함수(*method function*)라고 부른다. 객체지향기법에서는 자료와 동작을 다같이 중시하고 고찰하며 그 어느 하나도 우선시하지 않는다.

객체지향파라다임에서 자료와 작용을 동시에 고찰해야 한다는것은 정확치 않다. 계단식세련과 관련한 문제(5.1)로부터 자료가 더 중시되거나 작용이 더 중시될 때가 있다는 것은 명백하다. 그러나 객체지향파라다임과 관련한 개발단계에서는 자료와 작용이 마찬가지로 중요하다.

1장과 7장에서는 왜 객체지향파라다임이 구조화파라다임에 비하여 우월한가 하는데 대하여 고찰하였다. 이러한 리유의 기초에는 바로 잘 설계된 객체 즉 응집도가 강하고 결합도가 약한 객체는 하나의 물리적인 실체의 측면들을 모두 모형화한다는 사실이 놓여있다. 이것이 실현되는 세부는 숨겨 져 있다. 즉 객체와의 유일한 통신은 객체에 보내여지는 통보문을 통하여 진행된다. 결국 객체는 본질적으로 잘 정의된 대면부를 가진 독립적인 단위이다. 객체는 쉽게 안전하게 유지정비될수 있으며 회귀오유가 발생하는 기회도 감소되게 된다. 더우기 8장에서 설명하는바와 같이 객체는 재리용가능하며 이러한 재리용가능성은 바로 계승의 특성에 의해서 강화되게 된다. 이제 객체를 리용하여 개발하는 문제를 생각해 보자. 쏘프트웨어에 대하여 이러한 근본적인 구축블로크들을 결합하여 큰 규모의 제품을 개발하는것이 구조화파라다임을 리용하는것보다 더 안전하다. 객체는 본질적으로 제품의 독립적인 구성부분이기때문에 개발의 관리는 물론 제품의 개발이 더 쉬워 지며 그로 인하여 오유가 더 적게 발생되는것 같다.

객체지향파라다임의 우월성과 관련한 모든 측면은 한가지 문제를 제기한다. 즉 만일 구조화파라다임이 객체지향파라다임보다 나쁘다면 어째서 구조화파라다임이 그렇게 많이 성공할수 있었는가 하는것이다. 이것은 쏘프트웨어공학이 현실에 널리 도입되지 않았을 때에 구조화파라다임이 적용되였다는것을 리해하는것으로써 설명할수 있다. 대신 쏘프트웨어는 간단히 《작성》되였다. 관리자의 견지에서 보면 가장 중요한것은 프로그람작성자가 많은 코드행을 작성하는것이였다. 제품의 요구사항확정과 명세작성(체계분석)에는 거의나 품을 들이지 않았으며 설계는 전혀 진행할수 없었다. 구성 및 수정모형(3.1)은 1970년대에 전형적인 기법이였다. 대부분의 개발자들은 맨 처음에 방법상 구조화파라다임을 리용하였다. 이상하게도 그때 구조화기법은 쏘프트웨어산업에서 주요한 개선을 가져 왔다. 그러나 쏘프트웨어제품이 크기가 커짐에 따라 구조화기법이 불충분하다는것이 인식되기 시작했고 그에 대한 대책으로서 객체지향파라다임이 제안되였다.

이것은 또 다른 질문을 제기한다. 즉 객체지향파라다임이 다른 모든 현재의 기법들에 비하여 더 우월하다는것을 어떻게 확신할수 있는가? 객체지향기법이 현재의 기타 다른 기법보다 더 좋다고 하는데 대하여 증명할수 있는 파학적인 자료는 없다. 그러한 자료를 얻을수도 없다. 단지 객체지향파라다임을 적용해 본 회사들의 경험을 믿는것밖에는 다른 길이 없다. 어쨌든 객체지향파라다임을 리용하는것은 현명한 처사로 된다.

실례로 IBM은 객체지향기법을 리용하여 개발한 세개의 전혀 다른 프로젝트에 대하여 보고하였다[Capper, Colgate, Hunter, and James, 1994]. 거의 모든 측면에서 객체지향파라다임은 구조화파라다임을 훨씬 릉가하였다. 특히 발견된 오유의 수가 크게 줄어 들었으며 예측할수 없는 업무변경들을 초래하지 않은 개발과 유지정비단계에서의 변경요구

들이 훨씬 줄어 들었으며 적응 및 완전유지정비가능성이 크게 증가하였다. 그리고 비록 앞의 네가지 측면의 개선만큼 크지는 않고 또 성능상 뚜렷한 차이는 없다 할지라도 리용상의 측면에서도 개선이 있었다.

객체지향파라다임에 대한 개발자들의 태도를 결정하기 위하여 150명의 경험 있는 쏘프트웨어개발자들에 대한 조사가 진행되였다[Johnson, 2000]. 그들은 객체지향파라다임을 리용한 96명의 개발자와 여전히 고전적인 파라다임을 리용하여 쏘프트웨어제품을 개발하고 있는 54명의 개발자로 이루어 졌다. 객체지향파라다임그룹에 대한 긍정적인 태도가 훨씬 더 강하였다고 하여도 두 그룹이 모두 객체지향파라다임이 더 우월하다고 느꼈다. 두 그룹은 사실상 객체지향파라다임의 부족점을 무시하였다.

객체지향파라다임이 많은 우월성을 가짐에로 불구하고 일련의 난관들과 문제점들이 보고되였다. 제기된 문제들을 보면 흔히 개발로력과 크기와 관련되여 있다. 새로운것이 진행되는 첫 시기에는 그이후의 경우보다 더 오랜 시간이 걸린다. 즉 이러한 초기주기를 때때로 학습곡선(*learning curve*)으로 생각할수 있다. 그러나 객체지향파라다임이 어떤 개발기업체들에서 처음으로 리용될 때에는 학습곡선을 참고한다고 해도 예상했던것보다 더 오랜 시간이 걸린다. 이것은 구조화기법을 리용할 때보다 제품의 크기가 더 커지는 사정과 관련된다. 이러한 현상은 특히 제품이 도형사용자대면부(GUI)(10.3)를 리용할 때 더욱 현저하게 나타난다. 그후 사정은 크게 개선된다. 우선 유지정비비용이 더욱 낮아 졌고 제품의 전반적인 생명주기비용이 줄어 들었다. 둘째로 새 제품이 개발될 때 이미 개발된 프로젝트로부터 일부 클라스를 재리용하여 개발비용을 훨씬 더 줄일수 있었다. 이것은 특히 GUI를 처음으로 리용할 때 더욱 의의가 있다. 즉 GUI에 드는 많은 비용을 그다음 제품개발에서 보상 받을수 있게 된다.

계승문제는 해결하기가 더 힘들다. 계승을 리용하는 중요한 리유는 어미클라스와 약간 차이나는 새로운 부분클라스가 어미클라스 또는 계층도에 있는 임의의 다른 선조들에 영향을 주지 않고 만들어 질수 있다는것이다. 반대로 일단 제품이 실현되여 현재의 클라스에 변화가 생기면 계층도에서 그아래에 있는 모든 자손클라스에 영향을 주게 된다. 이것을 흔히 약한 기초클라스문제(*fragile base class problem*)라고 한다. 영향을 받은 부분은 적어도 다시 콤파일되여야 한다. 일부 경우에 관련이 있는 객체(영향을 받은 부분클라스의 실체들)의 방법들은 다시 코드작성되여야 하는데 이것은 간단치 않은 과제로 될수 있다. 이 문제를 최소화하기 위하여서는 개발과정에 모든 클라스들을 주의깊게 설계하는것이 중요하다. 이것은 클라스의 변화로 하여 초래되는 영향을 감소시키게 된다.

두번째 문제는 계승을 그릇되게 리용한 결과에 의하여 초래될수 있다. 명백하게 보호하지 않으면 부분클라스는 어미클라스들의 속성을 모두 계승하게 된다. 보통 부분클라스는 자기자신의 추가적인 속성들을 가지게 된다. 결과 계층도에서 보다 낮은 준위에 있는 객체들은 급속히 커져서 기억문제를 초래하게 된다[Bruegge, Blythe, Jackson, and Shufelt, 1992]. 이것을 극복하기 위한 한가지 방도는 《가능한껏 계승을 리용하라》라는 주장을 《적당한 곳에서 계승을 리용하라》하는 주장으로 바꾸어 고찰하는것이다. 이밖에 자손클라스에서 선조클라스의 속성이 필요 없으면 그 속성은 명백하게 배제되여야 한다.

세번째 문제는 다형성과 동적맺기로부터 제기된다. 이에 대하여서는 7.8에서 론의하였다. 한가지 마지막질문은 객체지향파라다임보다 더 좋은것이 언제인가는 있을

수 있겠는가 하는것이다. 지어 강력한 지지자들조차도 객체지향파라다임이 모든 쏘프트웨어공학문제에 대한 최종적인 대답으로 된다고 주장하지는 않는다. 더우기 오늘날 쏘프트웨어공학자들은 객체를 초월한 중요한 돌파구를 보고 있다. 결국 인간의 노력에 대해서 고찰할 때 과거의 발견이 현재 제시된것보다 더 우월하게 되는 분야는 거의나 없다. 객체지향파라다임이 장래의 방법론에 의하여 교체될것이라는것은 틀림 없다. 중요한 교훈은 현재의 지식에 기초하여 객체지향파라다임이 더 좋은것으로 교체될것이라는것이다.

요 약

이 장은 모듈에 대한 서술로 시작된다(7.1). 다음의 두 절은 모듈응집도와 결합도에 의하여 잘 설계된 모듈을 구성하는 방법을 해석하고 있다(7.2, 7.3). 특히 모듈은 강한 응집도와 낮은 결합도를 가져야 한다. 7.4부터 7.6까지에서 자료추상화와 처리절차추상화를 비롯하여 여러가지 형태의 추상화에 대하여 서술하였다. 자료교갑화에서(7.4) 하나의 모듈은 자료구조와 그 자료구조에서 수행되는 작용들을 포함한다. 추상자료형(7.5)은 그 류형의 실체에 대하여 실행되는 작용들과 함께 하나의 자료형으로 된다. 정보은페(7.6)는 실행세부를 다른 모듈에 숨기는 방법으로 모듈을 설계하는 내용으로 구성되여 있다. 추상화를 확장시켜 나가는 과정은 계승을 지원하는 추상자료형인 클라스에 대한 서술로 종결된다(7.7). 객체는 하나의 클라스의 실체이다. 다형성과 동적맺기는 7.8의 주제이며 객체의 응집도와 결합도는 7.9에서 서술하고 있다. 이 장은 객체지향파라다임에 대한 론의로 결속한다(7.10).

보 충

객체는 처음으로 문헌 [Dahl and Nygaard, 1996]에서 서술되였다. 이 장에서 취급하고 있는 많은 착상들은 처음에 파나스가 제기하였다[Parnas, 1971, 1972a, 1972b]. 쏘프트웨어개발에서 추상자료형을 리용하는 문제는 문헌 [Liskov and Zilles, 1974]에서 제기하였다. 이에 대한 또 하나의 주요한 론문으로는 문헌 [Guttag, 1977]이 있다.

응집도와 결합도에 대한 기본문헌은 문헌 [Stevens, Myers, and Constantine, 1974]이다. 합성/구조화설계의 착상은 객체로 확대되였다[Binkley and Schach, 1997].

객체에 대한 소개자료들은 문헌 [Meyer, 1997]에서 찾아 볼수 있다. 객체지향프로그람작성이 재리용을 촉진시킨 방법들은 문헌 [Meyer, 1987, 1990]에서 제시되였다. 서로 다른 류형의 계승에 대하여서는 문헌 [Meyer, 1996b]에서 제시되였다. 객체지향파라다임에 대한 여러가지 간단한 기사들은 문헌 [E1-Rewini et a1., 1995]에서 찾아 볼수 있다. 객체지향프로그람작성체계, 언어, 응용(OOPSL)에 관한 대회회보에는 넓은 분야의 연구론문들을 포함하고 있다. 그리고 회보부록에는 성과적으로 개발된 객체지향프로젝트들을 서술

한 보고서들이 포함되여 있다. IBM프로젝트에서 객체지향파라다임을 성공적으로 리용하였다는데 대하여서는 문헌 [Capper, Colgate, Hunter, and James, 1994]에서 서술하였다. 파라다임과 관련한 기타 경험에 대하여서는 *Communications of the ACM* 1995년 10월호에서 보고하였다. 문헌 [Johnson, 2000]에서는 객체지향파라다임에 대한 립장을 개괄하였다. 문헌 [Fayad, Tsai, and Fulghum, 1996]에서는 객체지향파라다임으로 어떻게 이전하였는가에 대하여 서술하고 있다. 여기서는 또한 관리자들의 충고도 많이 서술하고 있다. 객체지향척도에 대한 상세한 설명은 문헌 [Henderson-Sellers, 1996]에서 주었다. *IEEE Computer* 1992년 10월호에는《계약에 의한 설계》에 대하여 서술한 문헌 [Meyer, 1992a]을 비롯하여 객체에 대한 중요한 기사들이 포함되여 있다. 객체와 관련한 여러가지 기법들에 대하여서는 *IEEE Software* 1993년 1월호에서 찾아 볼수 있다. 여기서 이 분야의 주요 용어들을 정의한 스나이더(Snyder)의 론문 [Snyder, 1993]은 특히 가치가 있다. 다형성의 약점에 대하여는 문헌 [Ponder and Bush, 1994]에서 서술하고 있다. *Communications of the ACM* 1995년 1월호에는 객체기법에 관한 기사들을 제시하고 있다. *IBM Systems Journal* 1996년 2호도 이와 마찬가지다.

문　제

7.1. 당신이 익숙된 임의의 프로그람작성언어를 선택하시오. 7.1에서 준 모듈성에 대한 두개의 정의를 고찰하자. 두개의 정의가운데서 어느것이 선택한 언어로 모듈을 구성하는것을 직관적으로 리해할수 있게 하는 내용을 포함하고 있는가?

7.2. 다음의 모듈들에 대한 응집도를 결정하시오.

edit profit and tax record
edit profit record and tax record
read delivery record and check salary payments
compute the optimal cost using Aksen’s algorithm
measure vapor pressure and sound alarm if necessary

7.3. 당신은 제품개발에 참가하는 쏘프트웨어공학자이다. 당신의 관리자는 당신의 그룹이 설계한 모듈을 가능한껏 재리용할수 있게 하는 방법들을 조사할것을 요청한다. 그에게 무엇이라고 말하겠는가?

7.4. 당신의 관리자가 당신에게 현존 모듈들을 어떻게 재리용할수 있는가를 결정할것을 요청한다. 첫번째 과제는 일치적인 응집도를 가진 매개 모듈을 기능적인 응집도를 가진 개별적인 모듈로 분할하는것이다. 관리자는 개별적인 모듈들이 시험되지 않았거나 그것들이 문서화되지 않았다고 지적하였다. 당신은 이제 무엇이라고 말하겠는가?

7.5. 유지정비에 주는 응집도의 영향은 무엇인가?

7.6. 유지정비에 주는 결합도의 영향은 무엇인가?

7.7. 자료의 교갑화와 추상자료형을 정확히 구별하시오.

7.8. 추상화와 정보은페를 정확히 구별하시오.

7.9. 다형성과 동적맺기를 정확히 구별하시오.

7.10. 그림 7-23에 있는 설명문을 교원이 지적한대로 C++와 Java언어로 바꾸시오. 결과적인 모듈이 정확하게 실행된다는것을 확인하시오.

7.11. C++와 Java언어는 추상자료형의 실현을 지원하지만 그것은 정보은페를 포기하는 대가로 이루어 진다는것이 제기되였다. 이 주장을 론의하시오.

7.12. 이 장의 앞부분에 있는 《알고 싶은 문제》에서 지적한바와 같이 객체는 1966년에 처음으로 제기되였다. 거의 20년이 지나서 재발명된후에야 객체가 널리 쓰이게 되였다. 이 현상을 설명할수 있는가?

7.13. 교원이 구조화쏘프트웨어제품들을 배포해 줄것이다. 정보은페와 추상화의 준위, 결합도, 응집도의 관점에서 모듈을 분석하시오.

7.14. 교원이 객체지향쏘프트웨어제품을 배포해 줄것이다. 정보은페와 추상화의 준위, 결합도, 응집도의 관점에서 모듈을 분석하시오. 문제 7.13의 대답과 비교하시오.

7.15. (과정안상 목표) 부록 1에 있는 브로드랜즈지역아동병원제품이 구조화파라다임을 리용하여 개발되였다고 가정하자. 당신이 찾으려고 하는 기능적인 응집도로 된 모듈의 실례를 드시오. 이제 제품이 객체지향파라다임을 리용하여 개발되였다고 가정하자. 당신이 찾으려고 하는 클라스의 실례를 드시오.

7.16. (쏘프트웨어공학독본) 교원은 문헌 [Johnson, 2000]의 복사본을 배포해 줄것이다. 왜 응답자들이 객체지향파라다임에 대한 결함이 본질적으로 무의미하다고 간주한다고 생각하는가?

참 고 문 헌

[Berry, 1978] D. M. BERRY, personal communication, 1978.

[Binkley and Schach, 1996] A. B. BINKLEY AND S. R. SCHACH, "A Comparison of Sixteen Quality Metrics for Object-Oriented Design," *Information Processing Letters,* **57** (No. 6, June 1996), pp. 271–75.

[Binkley and Schach, 1997] A. B. BINKLEY AND S. R. SCHACH, "Toward a Unified Approach to Object-Oriented Coupling," *Proceedings of the 35th Annual ACM Southeast Conference,* Murfreesboro, TN, April 2–4, 1997, pp. 91–97.

[Blaha, Premerlani, and Rumbaugh, 1988] M. R. BLAHA, W. J. PREMERLANI, AND J. E. RUMBAUGH, "Relational Database Design Using an Object-Oriented Methodology," *Communications of the ACM* **31** (April 1988), pp. 414–27.

[Bruegge, Blythe, Jackson, and Shufelt, 1992] B. BRUEGGE, J. BLYTHE, J. JACKSON, AND J. SHUFELT, "Object-Oriented Modeling with OMT," *Proceedings of the Conference on Object-Oriented Programming, Languages, and Systems, OOPSLA '92, ACM SIGPLAN Notices* **27** (October 1992), pp. 359–376.

[Capper, Colgate, Hunter, and James, 1994] N. P. CAPPER, R. J. COLGATE, J. C. HUNTER, AND M. F. JAMES, "The Impact of Object-Oriented Technology on Software Quality: Three Case Histories," *IBM Systems Journal* **33** (No. 1, 1994), pp. 131–57.

[Chidamber and Kemerer, 1994] S. R. CHIDAMBER AND C. F. KEMERER, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering* **20** (June 1994), pp. 476–93.

[Dahl and Nygaard, 1966] O.-J. DAHL AND K. NYGAARD, "SIMULA—An ALGOL-Based Simulation Language," *Communications of the ACM* **9** (September 1966), pp. 671–78.

[El-Rewini et al., 1995] H. EL-REWINI, S. HAMILTON, Y.-P. SHAN, R. EARLE, S. MCGAUGHEY, A. HELAL, R. BADRACHALAM, A. CHIEN, A. GRIMSHAW, B. LEE, A. WADE, D. MORSE, A. ELMAGRAMID, E. PITOURA, R. BINDER, AND P. WEGNER, "Object Technology," *IEEE Computer* **28** (October 1995), pp. 58–72.

[Fayad, Tsai, and Fulghum, 1996] M. E. FAYAD, W.-T. TSAI, AND M. L. FULGHUM, "Transition to Object-Oriented Software Development," *Communications of the ACM* **39** (February 1996), pp. 108–21.

[Flanagan and Loukides, 1997] D. FLANAGAN AND M. LOUKIDES, *Java in a Nutshell: A Desktop Quick Reference,* 2nd ed., O'Reilly and Associates, Sebastopol, CA, 1997.

[Gerald and Wheatley, 1999] C. F. GERALD AND P. O. WHEATLEY, *Applied Numerical Analysis,* 6th ed., Addison-Wesley, Reading, MA, 1999.

[Goldberg and Robson, 1989] A. GOLDBERG AND D. ROBSON, *Smalltalk-80: The Language,* Addison-Wesley, Reading, MA, 1989.

[Guttag, 1977] J. GUTTAG, "Abstract Data Types and the Development of Data Structures," *Communications of the ACM* **20** (June 1977), pp. 396–404.

[Henderson-Sellers, 1996] B. HENDERSON-SELLERS, *Object-Oriented Metrics: Measures of Complexity,* Prentice Hall, Upper Saddle River, NJ, 1996.

[ISO/IEC 8652, 1995] "Programming Language Ada: Language and Standard Libraries," ISO/IEC 8652, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Johnson, 2000] R. A. JOHNSON, "The Ups and Downs of Object-Oriented System Development," *Communications of the ACM* **43** (October 2000), pp. 69–73.

[Knuth, 1974] D. E. KNUTH, "Structured Programming with **go to** Statements," *ACM Computing Surveys* **6** (December 1974), pp. 261–301.

[Liskov and Zilles, 1974] B. LISKOV AND S. ZILLES, "Programming with Abstract Data Types," *ACM SIGPLAN Notices* **9** (April 1974), pp. 50–59.

[Meyer, 1986] B. MEYER, "Genericity versus Inheritance," Proceedings of the Conference on

Object-Oriented Programming Systems, Languages and Applications, *ACM SIGPLAN Notices* **21** (November 1986), pp. 391–405.

[Meyer, 1990] B. MEYER, "Lessons from the Design of the Eiffel Libraries," *Communications of the ACM* **33** (September 1990), pp. 68–88.

[Meyer, 1992a] B. MEYER, "Applying 'Design by Contract'," *IEEE Computer* **25** (October 1992), pp. 40–51.

[Meyer, 1992b] B. MEYER, *Eiffel: The Language,* Prentice Hall, New York, 1992.

[Meyer, 1996b] B. MEYER, "The Many Faces of Inheritance: A Taxonomy of Taxonomy," *IEEE Computer* **29** (May 1996), pp. 105–8.

[Meyer, 1997] B. MEYER, *Object-Oriented Software Construction,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1997.

[Myers, 1978b] G. J. MYERS, *Composite/Structured Design,* Van Nostrand Reinhold, New York, 1978.

[Parnas, 1971] D. L. PARNAS, "Information Distribution Aspects of Design Methodology," *Proceedings of the IFIP Congress,* Ljubljana, Yugoslavia, 1971, pp. 339–44.

[Parnas, 1972a] D. L. PARNAS, "A Technique for Software Module Specification with Examples," *Communications of the ACM* **15** (May 1972), pp. 330–36.

[Parnas, 1972b] D. L. PARNAS, "On the Criteria to Be Used in Decomposing Systems into Modules," *Communications of the ACM* **15** (December 1972), pp. 1053–58.

[Ponder and Bush, 1994] C. PONDER AND B. BUSH, "Polymorphism Considered Harmful," *ACM SIGSOFT Software Engineering Notes* **19** (April 1994), pp. 35–38.

[Schach, 1996] S. R. SCHACH, "The Cohesion and Coupling of Objects," *Journal of Object-Oriented Programming* **8** (January 1996), pp. 48–50.

[Schach and Stevens-Guille, 1979] S. R. SCHACH AND P. D. STEVENS-GUILLE, "Two Aspects of Computer-Aided Design," *Transactions of the Royal Society of South Africa* **44** (Part 1, 1979), 123–26.

[Shneiderman and Mayer, 1975] B. SHNEIDERMAN AND R. MAYER, "Towards a Cognitive Model of Programmer Behavior," Technical Report TR-37, Indiana University, Bloomington, 1975.

[Snyder, 1993] A. SNYDER, "The Essence of Objects: Concepts and Terms," *IEEE Software* **10** (January 1993), pp. 31–42.

[Stevens, Myers, and Constantine, 1974] W. P. STEVENS, G. J. MYERS, AND L. L. CONSTANTINE, "Structured Design," *IBM Systems Journal* **13** (No. 2, 1974), pp. 115–39.

[Stroustrup, 1991] B. STROUSTRUP, *The C++ Programming Language,* 2nd ed., Addison-Wesley, Reading, MA, 1991.

[Yourdon and Constantine, 1979] E. YOURDON AND L. L. CONSTANTINE, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,* Prentice Hall, Englewood Cliffs, NJ, 1979.

제8장. 재리용성, 이식성, 호상조작성*)

수레바퀴를 재발명한것이 형법상 위반이라면 많은 쏘프트웨어전문가들은 감옥에서 옥고를 치르게 될것이다. 실례로 수만개(수십만개는 아니다)의 COBOL로임지불계산프로그람이 있는데 이것들은 모두 본질적으로 같은 일을 수행한다. 세계적으로 요구되는것은 바로 여러가지 하드웨어에서 동작할수 있고 잘 가공된 하나의 로임지불계산프로그람이다. 필요하다면 개별적인 기업체의 특정한 요구들에 응하는것이다. 그러나 이전에 개발된 로임지불계산프로그람대신에 자체의 로임지불계산프로그람을 령상태에서 개발하고 있다.

이 장에서는 왜 쏘프트웨어공학자들이 계속 《수레바퀴》를 재발명하고 있는가 그리고 재리용할수 있는 구성요소들을 리용하여 개발한 이식가능한 쏘프트웨어를 완성하기 위하여 무엇을 할수 있는가 하는것을 고찰한다. 우선 이식성과 재리용성을 구별하는것부터 론의를 진행한다.

8. 1. 재리용의 개념

령상태에서 제품을 다시 코드작성하는것보다 다른 콤파일러와 하드웨어조작체계상태에서 동작할수 있도록 전반적인 제품을 수정하는것이 더 쉽다고 하면 제품은 이식가능하다. 이와 대비적으로 재리용은 서로 다른 기능을 가진 서로 다른 제품개발을 촉진시키기 위하여 하나의 제품에 들어 있는 구성부분들을 리용하는것을 의미한다. 재리용가능한 구성부분은 반드시 하나의 모듈이나 코드토막일 필요는 없다. 즉 그것은 하나의 설계일수도 있고 지도서의 한 부분일수도 있으며 하나의 시험자료모임도 될수 있고 혹은 기간이나 비용타산일수도 있다.

재리용에는 두가지 류형이 있는데 하나는 우연적인 재리용이고 다른 하나는 계획적인 재리용이다. 새로운 제품을 개발하고 개발자가 이미 개발된 제품의 어느 한 구성부분을 새로운 제품에서 리용할수 있다면 이것을 우연적인 재리용 또는 기회적인 재리용이라고 한다. 쏘프트웨어의 구성부분을 리용하는것을 계획적인 재리용 혹은 체계적인 재리용이라고 한다. 우연적인 재리용에 비하여 계획적인 재리용의 잠재적인 우점의 하나는 앞으로 리용하기 위하여 특별히 개발한 구성부분들이 재리용하기 더 쉽고 안전하다는것이다. 즉 이러한 구성부분은 보통 로바스트적이며 잘 문서화되여 있고 철저히 시험된다. 이밖에 그 구성부분들을 유지정비하기 더 쉽도록 유일한 형태로 서술해 준다. 다른 한 측면은 회사안에서 계획적인 재리용을 실현하는데는 비용이 든다는것이다. 쏘프트웨어구성부분에 대하여 명세를 작성하고 설계, 실현, 시험, 문서작성하는데는 시간이 걸린다. 그러나 이러한 구성부분을 재리용함으로써 잠재적으로 재리용가능한 구성부분을 개발하는데

---

*) 앞에서 언급한바와 같이 이 장의 자료는 제2편에서 다시 고찰한다.

투자되는 비용이 보상된다는 담보는 없다.

처음에 콤퓨터가 개발되였을 때는 아무것도 재리용하지 못하였다. 매번 제품이 개발될 때마다 곱하기부분프로그람과 입출력프로그람, 시누스, 코시누스를 계산하는 부분프로그람과 같은 항목들을 령상태에서 개발하였다. 그러나 인차 이것은 로력랑비가 많다는것을 깨닫고 부분프로그람서고가 구축되였다. 그다음 프로그람작성자는 요구될 때마다 2차뿌리나 시누스함수를 간단히 불러 낼수 있었다. 이러한 부분프로그람서고는 더욱더 갱신되고 실시간지원부분프로그람으로 개발되였다. 따라서 프로그람작성자가 C++ 나 Java 방법을 호출할 때 정확히 관리하거나 인수를 넘길수 있도록 코드를 작성할 필요가 없게 되였다. 즉 적당한 실시간 지원부분프로그람을 호출함으로써 자동적으로 관리하게 되였다. 부분프로그람서고에 대한 개념은 SPSS[Norusis, 2000]와 같은 대규모통계서고와 NAG[Phillips, 1986]와 같은 수자처리분석서고들로 확장되였다. 클라스서고들은 객체지향언어를 리용하는데서 중요한 역할을 논다. Eiffel언어환경은 7개의 클라스서고들을 병합하였다[Meyer, 1990]. Smalltalk가 어느 정도 성공한것은 Smalltalk서고안에 아주 다양한 항목들이 있었기때문이다. 두 실례에서 모두 클라스서고를 찾아 보는것을 방조하는 CASE도구는 큰 도움을 주고 있다. C++와 관련하여 아주 많은 서로 다른 서고들이 리용되였다. 그 하나의 실례로는 C++표준서고(STL)가 있다[Musser and Saini, 1996].

응용프로그람작성대면부(API)는 보통 프로그람작성을 촉진시키는 조작체계호출들의 모임으로 된다. 실례로 Win32는 윈도우즈 2000과 윈도우즈NT와 같은 마이크로쏘프트조작체계를 위한 API이며 마킨토쉬도구통은 마킨토쉬조작체계 MacOS를 위한 API이다. 비록 API가 보통 조작체계호출들의 모임으로 실현된다고 하여도 API를 구성하는 부분프로그람은 프로그람작성자에게는 부분프로그람서고로 간주될수 있다. 실례로 Java응용프로그람작성대면부는 많은 프로그람묶음(서고)으로 구성되여 있다.

쏘프트웨어제품의 품질이 아무리 높다고 하여도 경쟁대상으로 되는 제품이 2년동안에 배포될수 있다고 할 때 시장에 나가는데 4년 걸린다고 하면 그것은 팔지 못하게 될것이다. 개발과정이 길게 되면 시장경제에서는 위험한것으로 된다. 《좋은》제품을 구성하는것과 관련한 모든 다른 기준은 그 제품이 시간적으로 경쟁할수 없다면 관계없다. 제품을 처음으로 출품하여 실패한 회사들에 대해서는 쏘프트웨어재리용기술이 유혹적인 기술로 되고 있다. 결국 현존구성부분이 재리용되면 그 구성부분을 다시 설계하고 실현하며 시험하고 문서작성할 필요가 없게 된다. 중요한것은 쏘프트웨어제품개발에서 평균 약 10%만이 원래의 목적을 실현하는데 리용되게 된다는것이다[Jones, 1984]. 리론적으로 제품의 다른 나머지 85%는 표준화되여 장래의 제품개발에 재리용될것이다.

85%라는것은 리론적으로 볼 때 재리용률에서 상한값으로 된다. 8.3에서 보여 준바와 같이 실천에서는 재리용률이 40%정도로 된다. 이로부터 다음의 질문을 제기된다. 만일 재리용률이 실천적으로 이 정도의 값을 가지게 되고 재리용이 결코 새로운 착상이 아니라면 왜 그렇게 적은 기업체들에서 개발과정을 단축하기 위하여 재리용을 진행하는가?

8. 2. 재리용의 장애

재리용하는데는 많은 장애들이 있게 된다. 즉

1. 많은 쏘프트웨어전문가들은 다른 전문가에 의해서 작성된 부분프로그람을 재리용하는것보다도 그 부분프로그람을 처음부터 다시 작성하려고 한다. 그것은 그들자신이 작성하지 않으면 부분프로그람이 좋다고 생각하지 않는다는것을 의미한다. 이것은 다른 말로 NIH(*not invented here*, 여기서 발명하지 않은)증후군으로 알려져 있다[Griss, 1993]. NIH는 관리상 문제이며 관리자측이 문제를 깨닫고 있으면 그 문제는 재리용을 촉진시키기 위하여 재정상 방조를 제공함으로써 해결될수 있게 된다.
2. 많은 개발자들은 부분프로그람이 제품에 오유를 발생시키지 않는다는것이 확실하면 그러한 부분프로그람을 선뜻 재리용하려고 할것이다. 쏘프트웨어의 품질과 관련하여 이러한 태도를 취하는것은 아주 당연한다. 결국 매 쏘프트웨어전문가들은 다른 사람이 작성한 쏘프트웨어에 오유가 있는것으로 본다. 여기서 문제해결방안은 그들이 재리용을 진행하기전에 철저히 시험을 진행하며 가능한껏 재리용부분프로그람에 복종하는것이다.
3. 큰 기업체들에서는 수십만개나 되는 잠재적으로 리용가능한 구성부분들을 가지고 있을수 있다. 이러한 구성부분들을 후에 다시 효과적으로 찾아 보자면 그것들을 어떻게 보관해야 하겠는가? 실례로 재리용가능한 구성부분자료기지는 2만개나 되는 항목들로 구성되고 여기서 정렬부분프로그람만 해도 125개나 될수 있다. 자료기지는 새 제품을 개발하는 개발자들로 하여금 125개의 정렬부분프로그람가운데서 어느것이 새 제품에 알맞는 부분프로그람인가를 인차 결정할수 있도록 구성되여야 한다. 보관과 검색문제를 해결하는것은 아주 다양한 문제해결을 제기하는 기술적인 문제로 된다[Meyer, 1987 or Prieto-Díaz, 1991].
4. 재리용은 비용이 든다. 트라즈(Tracz)는 다음의 세가지 측면에서 비용이 든다고 언급하였다. 즉 재리용가능한것을 만드는데 드는 비용과 그것을 재리용하는데 드는 비용, 재리용과정을 정의하고 실현하는데 드는 비용이 있다[Tracz, 1994]. 그는 재리용가능한 구성부분을 만드는데는 적어도 60%까지의 비용이 든다고 타산하였다. 일부 기업체들에서는 200%까지 비용이 든다고 보았고 지어는 480%까지도 비용이 들게 된다고 보고하였으며 반면에 8.5.3에서 보고한 어느 한 휼레트-패카드회사의 재리용프로젝트에서는 재리용가능한 구성부분들을 만들어 내는데 11%정도비용이 들었다고 보고하였다[Lim, 1994].

우에서 언급한 네가지 장애문제는 원리적으로 극복할수 있다.

다섯번째 문제는 쏘프트웨어계약과 관련되는 법률상의 문제로서 보다 더 문제성이 있다. 의뢰자와 쏘프트웨어개발기업체사이에 작성한 계약에 따르면 쏘프트웨어제품은 의뢰자에게 속한다. 따라서 쏘프트웨어개발자들이 어느 한 의뢰자의 제품에 있는 구성부분을 다른 의뢰자의 새로운 제품에 재리용하면 이것은 본질적으로 첫번째 의뢰자의 저작권에 대한 위반으로 된다. 개발자와 의뢰자가 같은 기업체의 성원들일 때는 내부의 쏘프트웨어에 대하여 이런 문제가 제기되지 않는다. 여섯번째 장애는 상업적인 규격(COTS) 구

성부분을 재리용할 때 제기되게 된다. 드문히 개발자들에게는 COTS구성부분의 원천코드가 주어 지게 되는 경우가 있게 된다. 그리하여 COTS구성부분을 재리용한 쏘프트웨어는 확장하고 수정하는데서 제한을 받는다.

알고 싶은 문제

WWW는 《도시신화(*urban myths*)》 즉 명백히 사실인 이야기들의 주요원천으로 되는데 이러한 이야기들은 암만해도 그 출처를 철저히 조사할수 없다. 이와 같이 한가지 도시신화는 코드의 재리용과 관련되여 있다.

이 이야기는 오스트랄리아공군이 직승기전투훈련을 위하여 가상현실감이 있는 훈련모의기를 설치하는것을 내용으로 하고 있다. 가능한껏 현실성 있게 대본을 만들기 위하여 프로그람작성자들은 자세한 풍경과 그리고(북부지방에 있는) 캉가루무리를 포함시켰다. 결국 직승기에 놀란 캉가루무리에 의하여 일어 난 먼지는 적들이 그 직승기의 위치를 발견할수 있게 하였다.

프로그람작성자는 캉가루의 움직임과 직승기에 대한 캉가루들의 반응을 모두 모형화할데 대한 지시를 받았다. 시간을 절약하기 위하여 프로그람작성자들은 원래 직승기의 엄호하에 공격하는 보병들의 동작을 모의하는데 리용하였던 코드를 재리용하였다. 여기서 다만 두개의 변경을 진행하였다. 즉 그들은 그림을 병사들로부터 캉가루로 바꾸었다. 그리고 그림의 이동속도를 빠르게 하였다.

어느 날 오스트랄리아비행사들의 한 그룹은 방문하여 온 미군비행사들에게 이 모의기를 가지고 진행하는 훈련동작들을 보여 주기로 하였다. 모의기는 가상적인 캉가루들를 윙윙 날아 다녔다. 기대했던바대로 캉가루들이 흩어 졌다가 언덕너머에서 다시 나타나 직승기에 스팅거미싸일을 발사하였다. 프로그람작성자들은 가상적인 보병들의 동작을 재리용할 때 그 코드부분을 제거해 버리는것을 잊어 버렸다.

그러나 모험집에서 서술한바와 같이 이 이야기는 전혀 도시신화가 아니라 훨씬 더 실제적인것 같다[Green, 2000]. 오스트랄리아국방과학 및 기술기업체에 있는 모의상륙작전과 책임자인 아네마리에 그리소고노(Anne-Marie Grisogono)박사는 1999년 5월 6일 오스트랄리아 캔베라에서 있은 회견에서 이 이야기를 말해 주었다. 비록 모의기가 가능한껏 현실감있게 설계되였다(공중사진을 나타내도록 지어 200만그루이상되는 나무를 포함시켰다.)고 하여도 캉가루는 장난삼아 포함시켰다. 프로그람작성자들은 직승기가 도착하였다는것을 캉가루가 발견할수 있도록 실지 스팅거미싸일발사과정을 재리용하였으며 직승기가 도착하면 정확히 캉가루가 무서워서 도망치도록 하기 위하여 캉가루가 뒤로 물러 나도록 동작을 설정하였다. 그러나 쏘프트웨어개발팀이 연구소에서 그 모의기를 시험하였을 때 그들은 무기와 불 붙는 동작을 다 제거해 버리는것을 잊어 버렸다. 또한 그들은 무기가 모의된 형태로 리용된다는것을 서술하지 않았으며 따라서 캉가루는 직승기에 발사할 때 여러가지 색갈의 큰 고무공들이 나타나는 암묵적인 무기를 사용하였다.

그리소고노는 캉가루가 즉시에 무장해제되였으므로 이제는 오스트랄리아상공을 안전하게 날아 다닐수 있게 되였다고 주장하였다. 쏘프트웨어전문가들은 재리용하는 코드가 너무 많이 재리용되지 않도록 여전히 주의를 돌려야 한다.

그러므로 법률상의 문제나 그리고 COTS구성부분을 리용하는 문제외에는 쏘프트웨어 기업체안에서 재리용과 관련하여 다른 주요한 장애가 없다(우의 《알고 싶은 문제》를 보시오.).

8．3．재리용의 실례연구

다음의 여섯가지 실례연구는 재리용이 실천에서 어떻게 성과를 거두었는가를 보여 주고 있다. 그것은 1976년부터 2000년까지 25년동안에 걸쳐 진행된것들이다.

8．3．1．레이손미싸일체계관리국

1976년에 레이손미싸일체계관리국은 설계와 코드에 대하여 계획적인 재리용이 가능한가 하는 연구를 진행하였다[Lanergan and Grasso, 1984]. 리용중에 있는 5,000개가 넘는 제품들을 분석하고 분류하였다. 연구자들은 업무응용제품에서 다만 6개의 기본적인 작용들만이 진행될수 있다는것을 결정하였다. 결과 40%부터 60%사이의 업무응용제품에 대하여 설계와 모듈이 표준화되고 재리용되였다. 기본적인 작용으로서는 자료의 정돈, 자료의 편집 또는 조작, 자료결합, 자료의 뒤집기, 자료갱신, 자료에 대한 보고서작성이 있다. 그 다음 6년동안은 가능한껏 설계와 코드를 재리용하기 위하여 노력하였다.

레이손연구는 두가지 방법으로 재리용을 진행하였다. 연구자들은 이 두가지 방법을 기능모듈(*functional modules*)과 COBOL프로그람론리구조라고 명명하였다. 레이손학술용어에서 기능모듈은 편집부분프로그람이나 자료기지처리절차부분호출, 세금계산부분프로그람, 접수된 계산서를 위한 날자시효계산부분프로그람과 같이 특정한 목적으로 설계되고 코드작성된 COBOL코드토막이다. 3,200여개의 재리용가능한 모듈들가운데서 60%의 재리용코드가 응용프로그람들에 적용되였다. 기능모듈은 주의깊게 설계되고 시험되며 문서작성된다. 이러한 기능모듈들을 리용한 제품들은 보다 더 믿음성 있고 전체적인 제품에 대하여 요구되는 시험을 더 적게 진행할수 있다는것을 알수 있었다.

모듈은 표준적인 복사서고에 보관되며 복사하여 얻을수 있다. 즉 코드는 응용프로그람제품에서 물리적으로 존재하는것이 아니라 콤파일할 때 COBOL콤파일러에 의해서 제품에 포함되게 된다. 이 방식은 C언어나 C++언어에서 **#include**와 류사하다. 그러므로 결과적인 제품의 원천코드는 복사된 코드가 물리적으로 존재하면 보다 더 짧아 지게 된다. 결과 유지정비는 보다 더 쉬워 진다. 레이손연구자들은 또한 COBOL프로그람론리구조라는것을 리용하였다. 이것은 완성된 제품에 대하여 더욱 보충되여야 할 기본틀거리이다. 론리구조의 한가지 실례는 갱신론리구조이다. 이것은 5.1.1에 있는 실례연구에서와 같이 련속적으로 갱신을 진행하는데 리용된다. 오유조종은 순차적인 검사과정이므로 갱신론리구조에 속한다. 론리구조는 길이로 22개의 단락이다(단락은 COBOL프로그람의 단위이다). get_transaction, print_page_headings, print_control_totals와 같은 기능모듈을 리용하여 많은 단락들이 채워 지게 된다. 그림 8-1에서는 기능모듈로 채워 놓은 단락들로 이루어 진 COBOL프로그람론리구조의 기본틀거리에 대하여 기호적으로 표현하고 있다.

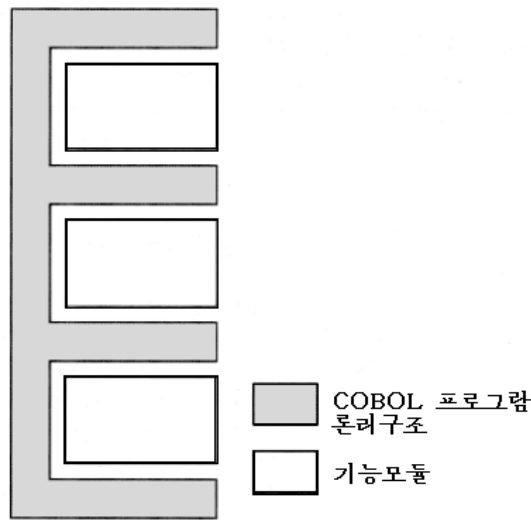



그림 8-1. 레이손미싸일체계관리국 재리용기구에 대한 기호적인 표현

이러한 제품에 대한 기본틀거리가 이미 만들어 져 있기때문에 제품의 설계와 코드작성은 보다 더 쉽고 빨리 진행할수 있게 된다. 여기서 필요한것은 상세하게 모두 채워 놓는것이다. 파일의 끝조건과 같은 오유를 범하기 쉬운 령역은 이미 시험되였다. 사실상 전체적인 시험도 더 쉬워 진다. 그러나 레이손연구자들은 주요한 우점이 바로 수정이나 확장에 있다고 보았다. 일단 유지정비프로그람작성자가 관련 있는 론리구조에 익숙하면 그는 거의나 원래의 개발팀성원처럼 된다.

1983년까지 론리구조는 새로운 제품을 개발하는데 5,500번이상 리용되였다. 코드의 약 60%는 기능모듈 즉 재리용가능한 코드로 구성되였다. 이것은 설계, 코드작성, 모듈시험, 문서작성시간이 60%까지 감소되며 쏘프트웨어제품개발의 생산성에 있어서 50%까지 증가된다는것을 의미한다. 그러나 레이손에 있어서 이러한 기법의 실제적인 리득은 규형의 일치로 인한 가독성과 피해가능성이 유지정비의 비용을 60-80%사이에서 줄일것이라는 기대에 있었다. 유감스럽게도 레이손은 필요한 유지정비자료를 얻을수 있기전에 자기일을 마쳤다.

재리용은 업무자료처리응용프로그람들에서만 적용될수 있는것 같다. 그러나 두번째 실례연구 즉 토시바쏘프트웨어에서 보여 준바와 같이 그렇지는 않다.

8. 3. 2. 토시바쏘프트웨어제작소

1977년에 토시바회사는 도꾜토시바휴츄공장에서 휴츄쏘프트제작을 시작하였다. 휴츄공장에서는 전력망, 원자력발전소, 자동차제작 등 여러 부분들에 필요한 산업공정조종체계를 생산하고 있으며 쏘프트웨어제작소에서는 그러한 체계들에서 요구되는 공정조종콤퓨터들을 위한 응용쏘프트웨어를 개발하고 있다[Matsumoto, 1984, 1987].

1985년까지 쏘프트웨어제작소에는 모두 2, 300명의 기술 및 관리성원들이 종사하고 있었다. 코드의 약 60%는 실시간부분프로그람에 의하여 담보되는 FORTRAN언어로 작성되여 있다. 그리고 20%는 아쎔블리어와 같은 언어들로 작성되고 기타 나머지는 문제에 따르는 해당한 언어로 작성되였다. 쏘프트웨어제작소는 코드의 행수로 생산성을 측정한다. FORTRAN언어로 1,000행 되는 코드를 작성하는데 드는 품은 아쎔블리어로 1,000행되는 코드를 작성하는데 드는 품과 차이나기때문에 생산성평가를 위한 단위를 등가인 아쎔블리어원천코드행수(EASL)로 설정하였다[Jones, 1996]. 여기서 일반적으로 고급언어로 작성된 코드 한행은 아쎔블리어에서는 4개 행에 해당된다고 보고 변화비례결수를 설정하였다. 이러한 측정방법을 리용하여 1985년에 쏘프트웨어제작소에서 개발된 제품은 720만 EASL이였다. 제품은 크기가 1백만EASL부터 2천 1백만EASL까지 있었는데 평균크기는 4백만EASL이였다.

쏘프트웨어는 폭포모형을 리용하여 개발되였으며 매 개발단계의 마감에서 자세한 심사와 검토를 진행하였다. EASL로 측정한 생산성은 쏘프트웨어개발을 추동하였다. 생산성은 프로젝트전반과 개별적인 프로젝트의 기초우에서 관리되였다. 제작소적으로 매해 생산성은 8～9%까지 올라 갔다. 성능평가에서 측정하여야 할 항목중의 하나는 개별적인 오유률이다. 실례로 프로그람작성자의 경우에 1,000EASL당 오유수는 숙련과 경험시간에 따라 감소되리라고 본다. 품질은 제작소에서 가장 중요한 측면으로 되고 있으며 심사와 검사, 품질토의를 포함한 많은 다른 기구를 통하여 품질이 보장되게 된다.

마쯔모도(Matsumoto)는 현존 쏘프트웨어를 재리용(우연적인 재리용)함으로써 생산성과 품질에서 모두 개선이 있었다고 하였다[Matsumoto, 1987]. 이러한 재리용가능한 쏘프트웨어는 모듈뿐아니라 설계, 명세서, 계약, 지도서와 같은 모든 종류의 문서들을 포함하게 된다. 위원회는 어느 구성부분을 재리용가능한 구성부분자료기지에 넣어 주는가를 결정한다. 여기서 자료기지에는 이후의 검색을 위하여 색인을 하였다. 자료기지에 있는 매 구성부분들에 대하여 재리용률을 통계낸다. 문서작성에 대한 재리용률은 재리용된 문서의 페지수를 총 페지수로 나눈것인데 1985년에 문서작성의 재리용률은 32%였다. 설계단계에서는 재리용률이 33%였고 실현단계에서는 코드의 48%가 재리용되였다. 더우기 재리용된 쏘프트웨어구성부분의 크기에 대하여 통계를 냈는데 약 55%는 크기가 1K-10K EASL였으며 36%는 10K-100KEASL범위에 있었다.

NASA에서 개발한 25개 쏘프트웨어제품에 대한 통계는 다음 부분에서 고찰하도록 한다.

8．3．3．NASA쏘프트웨어

쎌비(Selby)는 무인우주비행선조종을 위한 쏘프트웨어를 생산하는 NASA쏘프트웨어제품생산그룹에서 진행한 쏘프트웨어의 우연적인 재리용에 대하여 서술하였다[Selby, 1989]. 모두 25개의 쏘프트웨어들에 대하여 조사를 진행하였다. 그 쏘프트웨어제품들은 크기가 원천코드로 3,000~11,200행 되였다. 7,188개의 구성부분모듈에 대하여 네부류로 분류하였다. 그룹 1은 변화가 없이 리용된 모듈들이다. 그룹 2는 약간 수정한 모듈들이다. 즉 코드에 대하여 25%이하 변화시킨것들이다. 그룹 3은 25%이상의 코드를 수정한 모듈

들로 구성된다. 그룹 4는 처음부터 개발한 모듈들이다.

표본 실례에 있는 2,954개의 전체 FORTRAN모듈들이 자세히 연구되였다. 구체적으로 말하면 28%는 그룹 1에, 10%는 그룹 2에, 7%가 그룹 3에 속한다. 일반적으로 재리용된 모듈들은 작고 잘 문서화되여 있다. 이런 모듈들은 간단한 대면부와 약간한 입출력처리를 진행하는것들이며(그림 7-7에서와 같이) 모듈호상접속도에서 말단마디쪽에 있다.

이 결과들은 사실 놀라운것은 아니다. 작고 잘 문서화된 모듈들은 문서화가 힘든 큰 모듈들에 비하여 리해하기가 더 쉬우며 따라서 재리용되기가 더 쉽다. 더우기 큰 모듈들은 특정한 작용보다도 많은 다른 작용들을 수행할수 있다. 따라서 보다 작은 모듈들보다 더 적게 재리용될수 있다. 복잡한 대면부는 인수들이 아주 많다는것을 암시하는데 이것은 모듈의 재리용률을 떨어 뜨리는 경향이 있다. 입출력처리는 어느 정도 응용프로그람에 특정하며 더 적게 재리용될수 있다. 마지막으로 모듈호상접속도의 말단에 있는 모듈들을 그우에 있는 모듈들과 비교해 보면 말단모듈들은 보다 더 특정한 과제를 수행할수 있으며 반면에 웃쪽에 있는 모듈들은 결심에 따른다고 본다(이에 대하여 15.1.1에서 깊이 론의되였다.). 결과 말단모듈은 그밖의 모듈보다 더 재리용가능하게 된다.

쎌비의 결과를 고찰하는데서 건설적인 방법은 모듈들을 앞으로의 제품개발에서 재리용할수 있다는것이다. 관리자는 특정한 설계객체는 간단한 대면부를 가진 작은 모듈이여야 한다는것을 담보하여야 한다. 입출력처리는 몇개의 모듈에 국한되여 있다. 모든 모듈들은 철저히 문서화되여야 한다.

NASA그룹과 휴츄쏘프트웨어제작소사이에는 중요한 차이가 있다. 특히 쏘프트웨어를 재리용할데 대한 결심은 NASA쏘프트웨어개발자들의 독단적인 선택이였다. 즉 그 어떤 관리상 지시가 없었다. NASA성원들은 재리용이 가치 있는 쏘프트웨어공학기술이라는것을 믿고 있었기때문에 간단히 쏘프트웨어를 재리용하였다. 이밖에 재리용과정을 방조하는 쏘프트웨어도구는 없다. 이러한 상황은 휴츄공장에서 진행하는 재리용지향의 관리와 명백하게 대조되며 거기서는 정교한 쏘프트웨어구성부분 검색기구를 리용하게 된다. 이럼에도 불구하고 놀랍게도 NASA에서는 높은 비률로 재리용을 진행하고 있다.

네번째 실례연구는 재리용에 대한 관리계약의 효과를 명백히 하고 있다.

8. 3. 4. GTE자료봉사기구

우연적인 재리용계획을 성공적으로 실현한것은 GTE자료봉사였다[Prieto-Díaz, 1991]. NASA 실례연구와는 달리 GTE계획에 있어서 중요한 측면은 원천코드모듈의 재리용에 대한 완전한 관리공약이였다. 재리용을 장려하기 위하여 모듈을 재리용하는데 50～100딸라의 현금을 보상해 주도록 하였는데 모듈이 실지로 재리용될 때 보상을 하였다. 더우기 관리자의 예산은 그들이 관리하는 프로젝트가 높은 수준에서 재리용되게 될 때 늘어 난다. 지어 달마다 보수를 받는 재사용자도 있다.

이 결과 다음과 같은 일이 있었다. 1988년에 재리용수준은 14%였다. 이것은 회사에서 타산한데 의하면 백만딸라를 들인것으로 된다. 그다음에는 재리용수준이 20%로 올라갔고 1993년에는 50%로 예견한다고 타산되였다. GTE가 재리용계획을 그렇게 강하게 밀

고 나간 리유는 그때까지 총체적으로 천만딸라이상 절약할것을 예견한것이다.

GTE 재리용계획은 많은 흥미 있는 측면들을 보여 주었다. 첫째로, 새로운 모듈이 추가되였다고 하더라도 재리용에 쓸수 있는 모듈의 총수는 1988년에 190개로부터 1990년에는 128개로 작아 졌다. 그것은 바로 GTE자료봉사기구와 같은 기업체들에서는 재리용가능한 구성부분들을 가지고 거대한 명세목록을 작성할 필요가 없다는것을 보여 주었다. 둘째로, 보다 많은 보상를 위하여 큰 모듈(1만개이상의 코드)에 대하여 관심을 둔것이다. 보다 작은 모듈들을 재리용하려고 한 NASA의 경험과 대비적으로 GTE는 큰 모듈들을 재리용하는데서 성공하였다. 이러한 차이는 임의의 재리용프로그람에 대하여 관리공약을 취하는것이 얼마나 중요한가를 강조하고 있다.

8. 3. 5. 휼레트-패카드회사

휼레트-패카드회사는 회사의 다른 여러 국들에서 재리용프로그람을 실현하였다[Lim, 1994]. 일반적으로 이런 프로그람들은 재리용된 결과 쏘프트웨어의 품질이 개선되였다는 견지에서 성공적이였다. 여기서는 두개의 특성을 가진 프로그람을 론의하고 세번째는 8.5.4에서 고찰하기로 한다.

쏘프트웨어기술국의 제조업에 종사하는 생산과에서는 1983년부터 우연적인 재리용프로그람을 가지고 있었다. 이 과에서는 제조업에서의 지원계획작성에 필요한 쏘프트웨어를 개발하였다. 재리용하기 위하여 선택된 구성부분들은 Pascal과 SPL(HP3000콤퓨터체계상에서 동작하는 체계쏘프트웨어를 위한 언어)언어로 작성되였다. 새로운 코드에 대한 오유률은 1,000행되는 코드당(KLOC) 4.1개의 오유발생으로 된다. 그러나 재리용코드에 대하여서는 KLOC당 0.9개의 오유발생으로 된다. 재리용의 결과 전체적인 오유률은 KLOC당 2.1개의 오유로 떨어 졌으며 51% 감소된것으로 된다. 생산성은 1992년에는 57%로 늘어 났으며 월공수는 1.1KLOC까지 늘어 났다. 놀랍게도 그다음해에는 프로젝트가 승산이 없게 되였다.

1987년부터 휼레트-패카드회사의 싼디에고 기술도형(STG)국에서는 재리용프로그람을 계획하였다. 이 국에서는 작도기와 인쇄기를 위한 펌웨어(*firmware*)를 개발하고 유지정비한다. C언어로 2만행되는 작은 제품이 3년동안 개발되고 그다음 재리용되였다. 1987년과 1994년(1994년자료는 평가자료임)사이에 재리용프로그람에 드는 전체 비용은 160만딸라였고 560만딸라가 절약되였다. 오유률도 KLOC당 1.3개의 오유로 되여 24% 감소되였다. 또한 생산성은 월공수가 0.7KLOC로 40%까지 올라 갔다. 마지막으로 제품배포시간도 24%까지 줄어 들었다.

STG재리용프로그람개발에 드는 비용에도 흥미를 끌고 있다. 재리용가능한 펌웨어구성부분을 개발하는데 드는 비용은 류사한 재리용불가능한 구성부분을 개발하는데 드는 비용보다 다만 11% 더 들었다. 통합단계에 드는 비용은 재리용불가능한 구성부분을 개발하는데 드는 비용의 19%였다. 즉 구성부분이 매번 재리용될 때마다 드는 비용은 그 구성부분을 령상태에서 개발하는데 드는 비용의 약 1/5뿐이였다.

휼레트-패카드회사는 현재 다음과 같은 더 좋은 구상을 가지고 있다. 하나의 인쇄기모형으로부터 얻은 펌웨어를 그것에 련이은 모형에서 리용할 대신에 쏘프트웨어제품개발

지침을 구상하였다[Toft, Coleman and Ohta, 2000]. 이것에 대해서는 8.5.4에서 서술하였다.

이러한 5개의 실례연구에 대한 총체적인 교훈은 바로 재리용이 실천에서 가능하다는것이며 비용을 절약할수 있다는것이다. 그러나 관리자측이 재리용을 밀고 나가는것이 기본이다.

마지막 실례연구는 성공적이라고 하기보다는 경계해야 할 이야기이다.

8．3．6．유럽우주항공국

1996년 7월 4일에 유럽우주항공국은 아란 5호로케트를 처음으로 발사시켰다. 쏘프트웨어결함으로 로케트는 리륙한 다음 37s만에 떨어 졌다. 로케트와 탄두의 비용은 5억딸라였고 이것은 현재까지 비용이 가장 많이 든 쏘프트웨어고장이였다[Jézéguel and Meyer, 1997].

실패의 기본원인은 64bit옹근수를 16bit부호 없는 옹근수로 변환하려고 시도한데 있었다. 변환될 수는 2^{16}보다 더 컸다. 그리하여 Ada에서 **exception**(실행시 오유)이 발생하였다. 유감스럽게도 코드에는 이러한 오유를 처리하는 명확한 오유조종프로그람이 없었기때문에 쏘프트웨어가 폭주되였다. 이것은 한 기판콤퓨터를 마비시켰고 그다음 아란 5호로케트를 파괴시켰다.

사실 오유를 일으킨 변환은 불필요한것이였다. 리륙하기전에 관성체계를 바로 잡도록 정확하게 계산이 진행되였다. 이러한 계산은 리륙하기전 9s동안 정지되였다. 그러나 초읽기가 계속 진행되면 초읽기가 다시 시작된 다음에 관성체계를 재개시하는것은 몇시간동안 진행될수 있다. 그러한 현상을 막기 위하여 비행을 시작한 다음에 50s동안 계속 계산을 진행하여 비행을 잘하게 한다(그럼에도 불구하고 일단 리륙하면 관성체계를 조절하는 방도는 없게 된다.). 이런 조종과정이 쓸데없이 계속되여 실패를 일으켰다.

유럽우주항공국은 효률적인 쏘프트웨어품질보증구성부분을 병합한 쏘프트웨어개발공정을 리용하였다. 그러면 왜 Ada코드에 자리넘침과 같은 가능성을 조종하기 위한 례외조종프로그람이 없었는가? 콤퓨터에 지나친 부담을 주지 않기 위하여 자리넘침을 초래하지 않는다고 본 변환들을 대책이 없이 내버려 둔것이였다. 문제로 되는 코드는 10년전의것이였다. 그것은 아란 4호로케트를 조종하는 쏘프트웨어를 더 시험도 하지 않고 변경도 시키지 않은 채로 재리용하고 있었다. 수학적으로 분석한데 의하면 질문에서 제기된 계산은 아란 4호에서는 완전히 안전하였다는것이 증명되였다. 그러나 그 분석은 아란 4호에 대하여서는 정확하지만 아란 5호에 대하여서는 정확치 않다고 하는 확실한 가정에 기초하여 진행되였다. 따라서 분석은 더는 정확하지 못하였으며 코드는 자리넘침에 대한 가능성을 고려할수 있는 례외조종프로그람의 보호를 요구하였다. 성능상 제한은 없었지만 아란 5호의 Ada코드전반에 걸쳐 례외조종프로그람은 명백히 있었다. 만약 관련 있는 모듈에 변화되여야 할 수는 2^{16}보다 작아야 한다는 주장을 포함하고 있었더라면 시험시에나 그리고 제품이 설치된 다음에라도 **assert**명령을 리용하여 아란 5호를 파괴시키지 않았을수도 있었다[Jézéguel and Meyer, 1997].

재리용의 경험과 관련하여 주요한 교훈은 바로 하나의 문맥안에서 개발된 쏘프트웨어는 다른 문맥에서 재리용될 때 반드시 재시험되여야 한다는것이다. 즉 재리용된 쏘프

트웨어모듈은 그자체로서는 재시험할 필요는 없지만 재리용된 제품으로 통합된 다음에는 재시험되여야 한다. 또 하나의 교훈은 6.5.2에서 론의한바와 같이 수학적인 증명의 결과에 대하여 전적으로 믿는것은 현명하지 못하다는것이다.

다음에 재리용에 대한 객체지향파라다임의 영향에 대하여 검토한다.

8. 4. 객체와 재리용

합성-구조화설계(C/SD)리론이 약 30여년전에 처음으로 제기되였을 때 리상적인 모듈은 기능적인 응집도를 가지는 모듈이라는 주장이 있었다(7.2.6). 즉 모듈이 다만 한가지 작용을 수행하면 그것은 전형적인 재리용후보로 간주 되였으며 이와 같은 모듈의 유지정비는 쉬워 질것으로 예상되였다. 이러한 론법에서 결함은 기능적인 응집도를 가진 모듈은 자체포함되지도 않고 독립적이지도 않다는것이다. 대신 자료에 대하여서는 처리해야 한다. 만일 그러한 모듈이 재리용되면 처리하여야 할 자료도 재리용되여야 한다. 새 제품에 있는 자료가 원래의 제품에 있는 자료와 같지 않으면 자료가 변화되여야 하든가 또는 기능적인 응집도를 가진 모듈이 변화되여야 한다. 그러므로 흔히 믿군 하는것들과 반대되게 기능적인 응집도는 재리용에서는 리상적인것이 못된다.

고전적인 C/SD에 따르면 그다음 가장 좋은 류형의 모듈은 정보적인 응집도를 가진 모듈이다(7.2.7). 최근 그러한 모듈은 본질적으로 하나의 객체 즉 어떤 클라스의 실례라고 인정되고 있다. 잘 설계된 객체는 특정한 실세계의 실체에 대하여 모든 측면을 모형화하지만 자료와 그 자료를 처리하는 작용의 실현을 숨겨 버리기때문에 그것은 쏘프트웨어의 기본구성블로크로 된다. 이리하여 객체지향파라다임이 정확히 리용되면 결과로 되는 모듈(객체)은 정보적인 응집도를 가지며 이것은 재리용을 촉진시킨다.

8. 5. 설계 및 실현단계에서의 재리용

설계단계에서 아주 다른 형태의 재리용이 가능하다. 재리용되는 자료는 한개 또는 두개의 모듈로부터 완성된 쏘프트웨어제품의 구성에 이르기까지 다양할수 있다. 이제 여러가지 형태의 설계재리용과 일부 실현단계에서 진행되게 되는 재리용에 대하여 검토를 진행한다.

8. 5. 1. 설계의 재리용

제품을 설계할 때 설계팀 성원은 초기의 설계단계에서부터 모듈이나 또는 클라스가 현재의 프로젝트에서 리용될수 있도록 한다. 이런 형태의 재리용은 은행업무나 항공운수체계와 같은 특정한 응용령역에서 쏘프트웨어를 개발하는 기업체들에 대하여서는 특히 공통적이다. 기업체는 앞으로 재리용하기 위하여 설계요소들을 보관해 놓고 설계자들로 하여금 그것들을 재리용하도록 고무해줌으로써 이런 형태의 재리용을 촉진시킬수 있다. 이 형태의 재리용은 한계가 있지만 두가지 우점이 있다. 첫째로, 시험된 모듈설계는 제품

에 병합된다. 따라서 전반적인 설계는 처음부터 진행할 때보다 더 빨리 그리고 질이 더 좋게 작성될수 있다. 둘째로, 모듈의 설계를 재리용할수 있으면 그 모듈의 실현도 재리용할수 있다.

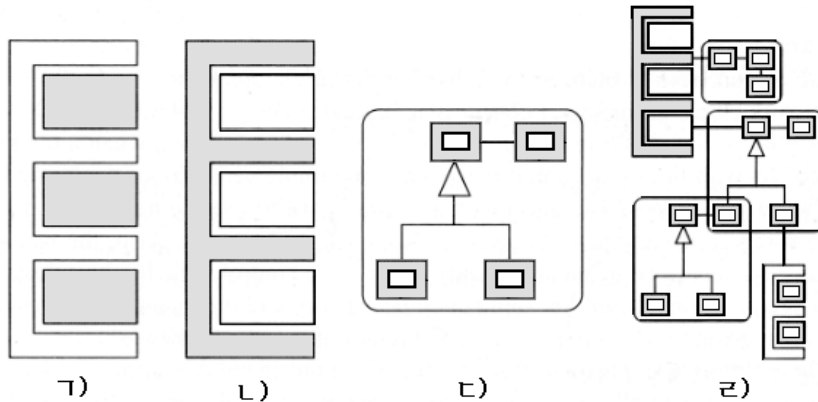



그림 8-2. 네가지 류형의 설계재리용
검은 부분은 각각 ㄱ) 서고나 도구틀, ㄴ) 틀거리, ㄷ) 설계패턴, ㄹ) 틀거리, 도구틀, 세 설계패턴을 포함한 쏘프트웨어구성에 있는 설계재리용을 의미한다.

이러한 방법은 그림 8-2의 ㄱ)에서 보여 준바와 같이 서고재리용으로 확장할수 있다. 서고는 관련 있는 재리용가능한 부분프로그람들의 모임이다. 실례로 쏘프트웨어개발자들은 행렬전위나 고유값탐색과 같은 일반적인 과제들을 수행할수 있는 부분프로그람을 작성한다. 대신 LAPACK2.0 (Anderson et al., 1995)과 같은 과학서고를 구입하게 된다. 그다음 필요할 때마다 과학서고에 있는 부분프로그람들은 앞으로의 쏘프트웨어에서 리용되게 된다. 객체지향파라다임에 대한 인기가 올라감에 따라 과학쏘프트웨어를 위한 LAPACK++ [Dongarra, Pozo, and Walker, 1993], DiffPack [Langtangen, 1994], C-XSC[Klatte et al., 1991] 등의 클라스서고들이 개발되였다.

또 하나의 실례로서는 도형사용자대면부(GUI)를 위한 서고가 있다. GUI방법들을 처음부터 작성할 대신에 GUI클라스서고나 도구틀 즉 GUI의 모든 측면을 조종할수 있는 클라스들의 모임을 리용하는것이 훨씬 더 편리하다. 이와 관련한 많은 GUI도구틀들이 있다 [Flanagan and Loukides, 1997].

서고를 재리용하는데서 문제는 흔히 서고들이 재리용가능한 설계가 아니라 재리용가능한 부분프로그람의 모임으로 되여 있다는것이다. 도구틀 역시 일반적으로 설계재리용이 아니라 코드재리용을 촉진시킨다. 객체지향파라다임을 리용할 때 이 문제는 열람기 즉 계층나무로 현시하는 CASE도구를 리용함으로써 완화될수 있다. 설계자는 그다음 계층나무로 된 서고를 자세히 살펴 보면서 여러가지 클라스의 항목을 조사하여 어느 클라스를 현재의 설계에 적용할수 있는가를 결정한다.

서고와 도구틀의 재리용과 관련한 중요한 측면은 그림 8-2의 ㄱ)에서 보여 준바와 같이 설계자가 제품의 전반적인 조종론리에 대하여 책임을 진다는것이다. 서고나 도구틀

은 쏘프트웨어개발공정들에 배포되여 제품의 특정한 조작을 병합하는 설계의 부분들을 제공해 준다.

다른 한편 응용프로그람의 기본틀거리는 조종론리를 제공해 주는 서고나 도구들을 반전한것이다. 즉 개발자들은 특정한 조작의 설계에 책임이 있다. 이에 대해서는 다음 부분에서 서술한다.

8. 5. 2. 응용프로그람의 틀거리

그림 8-2의 ㄴ)에서 보여 준바와 같이 응용프로그람틀거리(*application framework*)는 설계의 조종론리를 병합한다. 틀거리를 재리용할 때 개발자들은 개발되는 제품의 응용상 특정한 조작들에 대하여 설계를 진행해야 한다. 응용상 특정한 조작들이 삽입되는 곳을 흔히 지적점(*hot spot*)이라고 한다.

용어 틀거리(*framework*)는 최근에 보통 객체지향응용프로그람틀거리라고 부른다. 실례로 [Gamma, Helm, Johnson, and Vlisside, 1995]에서는 틀거리를《쏘프트웨어의 특정한 클라스에 대하여 재리용가능한 설계를 진행할수 있는 협조적인 클라스들의 모임》으로 정의하였다. 그러나 8.3.1에 있는 레이손미싸일체계관리국 실례연구를 고찰해 보자. 그림 8-1은 그림 8-2의 ㄴ)와 동등하다. 달리 말하면 1970년대의 레이손 COBOL프로그람론리구조는 오늘날의 객체지향응용프로그람틀거리에 대한 고전적인 조상이다.

응용프로그람틀거리의 한가지 실례는 콤파일러의 설계를 위한 클라스의 모임이다. 설계팀은 언어와 그리고 목적하는 기계에 특정한 클라스들을 제공해 주어야 한다. 그다음 이러한 클라스들은 그림 8-2의 ㄴ)에서 흰칸으로 보여 준바와 같이 틀거리에 삽입된다. 틀거리에 대한 또 하나의 실례는 ATM을 조종하는 쏘프트웨어를 위한 클라스들의 모임이다. 여기서 설계자들은 은행업무망의 ATM에 의하여 제공되는 특정한 은행업무봉사를 위한 클라스들을 제공해 주어야 한다.

틀거리를 재리용하면 두가지 리유로 하여 도구들을 재리용할 때보다 제품개발을 더 빨리 진행할수 있도록 한다. 첫째로, 설계가 틀거리를 리용하여 재리용되면 될수록 처음부터 설계를 진행하게 되는 경우에는 그만큼 더 적어 지게 된다. 둘째로, 틀거리(조종론리)를 리용하여 재리용되는 설계부분이 조작보다도 설계하기가 더 힘들기때문에 도구들을 재리용할 때보다 결과적인 설계의 품질도 더 좋아 지게 된다. 서고나 도구들을 리용하여 재리용할 때 틀거리의 실현도 역시 재리용될수 있다. 개발자들은 틀거리들에 대한 이름과 호출약속을 리용해야 하지만 그 비용은 적다. 또한 조종론리가 응용프로그람틀거리를 재리용하는 기타 다른 제품들에서 시험되고 유지정비자들이 이미 그것과 같은 틀거리를 재리용하는 또 다른 하나의 제품을 유지정비할수 있기때문에 결과적인 제품은 쉽게 유지정비되게 된다.

응용프로그람틀거리를 내놓고도 많은 코드틀거리들이 있다. 처음으로 성공한 코드틀거리중에는 마킨토쉬상에서 동작하는 응용쏘프트웨어를 개발하는 틀거리인 MacApp가 있다[Wilson, Rosenstein, and Shafer, 1990]. 마이크로쏘프트기초클라스서고(MFC)는 Windows 응용프로그람에서 GUI를 구축하도록 하는 틀거리들의 큰 집합이다. MFC응용프로그람은 창문의 이동과 크기조절 그리고 대화창을 리용한 입력처리, 마우스를 찰칵하는것과 같은

사건처리라든가 차림표선택과 같은 사건처리 등과 같은 표준창문조작들을 진행할수 있다[Holzner, 1993]. Object Windows Library (OWL)을 갱신한 Borlands Visual Component Library (VCL)도 기능상 MFC와 류사하다. 그러나 VCL은 완전히 객체지향적이다. 이것이 바로 VCL이 MFC보다 우월하다고 보는 많은 리유들중의 하나이다[Wells, 1996].

이제는 설계패턴에 대하여 보기로 하자.

8. 5. 3. 설계패턴

크리스터퍼 알렉싼더(다음의 《알고 싶은 문제》를 보시오.)는 다음과 같이 말했다. 《매개 패턴은 우리의 주위환경에서 자주 일어 나는 문제들을 표현해 준다. 같은 방법으로 두번 진행하지는 않고 백만번이상 리용할수 있는 그러한 방법으로 문제해결의 핵을 나타낸다.》[Alexander et al., 1977] 그는 비록 건물이나 기타 다른 구조적인 객체에 대한 문맥에서 서술하고 있지만 이 견해는 다 설계패턴에 마찬가지로 응용할수 있다.

설계패턴은 특정한 설계를 창조하도록 주문작성해야 하는 호상작용하는 클라스들의 모임형태로 일반적인 설계문제를 풀어 나가는 해결책으로 된다. 이에 대하여서는 그림 8-2의 ㄷ)에서 보여 주었다. 선들로 련결된 컴컴한 칸들은 호상작용하는 클라스들을 의미한다. 컴컴한 칸에 있는 흰칸들은 이러한 클라스들이 특정한 설계를 위하여 만들어 져야 한다는것을 의미한다.

알고 싶은 문제

객체지향쏘프트웨어공학분야에서 가장 영향력 있는 사람들중의 한 사람은 크리스토퍼 알렉싼더인데 그는 객체나 쏘프트웨어공학에 대하여 거의나 아무것도 모르는 세계적으로 유명한 건축설계가이다. 그는 책[Alexander et al., 1977]에서 도시나 건물, 방들과 정원 등을 서술하는 건축을 위한 패턴언어들을 서술하였다. 그의 착상을 쏘프트웨어공학자들 특히 소위 4인그룹[Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides]을 받아 들여 적용하였다. 설계패턴에 관해서 가장 잘 팔리는 책[Gamma, Helm, Johnson, and Vlissides, 1995]은 객체지향집단이 널리 받아 들인 알렉싼더의 구상을 반영하였다.

패턴은 기타 다른 분야들에서도 제기된다. 실례로 비행장에 착륙할 때 비행조종사는 정확한 착륙패턴 즉 방향이나 고도 등을 알아야 한다. 또한 옷재단에서 패턴은 특정한 옷들을 만드는데 반복적으로 리용될수 있는 형태들이다. 패턴의 개념은 그자체로서는 의미가 없다. 중요한것은 쏘프트웨어개발과 특히 설계에 패턴을 응용하는것이다.

패턴들이 어떻게 쏘프트웨어개발을 방조하는가를 리해하기 위하여 다음의 실례를 고찰해 보자. 이제 어느 한 쏘프트웨어기업체가 부분품생성프로그람 즉 도형사용자대면부를 구축하는데 방조를 줄수 있는 도구를 개발하려고 한다고 하자. 여러가지 도구(창문이나 단추, 차림표, 화면흐름띠와 같은)들을 령상태에서 개발해야 할 대신 개발자들은 응용프로그람안에서 리용되는 부분품들을 정의하고 있는 부분품생성프로그람에 의하여 만들어 진 클라스들의 모임을 리용할수 있다.

문제는 응용프로그람이 Linux, Mac OS, Windows를 비롯한 많은 서로 다른 조작체계상에서 동작할수 있도록 하는것이다. 부분품생성프로그람은 이 세가지 조작체계를 지원하는것이다. 그러나 부분품생성프로그람이 응용프로그람에 대하여 하나의 특정한 체계상에서 동작하는 부분프로그람을 힘들게 코드작성하면 앞으로 그 응용프로그람을 수정하기가 힘들게 되고 생성된 부분프로그람을 다른 조작체계상에서 동작할수 있는 다른 부분프로그람으로 교체한다는것도 힘들게 될것이다. 실례로 응용프로그람이 Linux에서 동작할 예정이라고 하자. 그러면 차림표가 생성될 때마다 매번 통보문 create Linux menu가 보내진다. 그러나 이제는 응용프로그람이 Mac OS상에서 동작하여야 한다. create Linux menu의 매 실체들이 create Mac OS menu로 교체되여야 한다. 큰 응용프로그람에 대하여 Linux에서 Mac OS로 변화시키는것은 힘들고 오유를 범하기 쉽다.

해결책은 바로 응용프로그람을 특정한 조작체계와 떼여 놓는 방법으로 부분품생성프로그람을 설계하는것이다. 이것은 설계패턴추상제작소(*Abstract Factory*)를 리용하여 할수 있다[Gamma, Helm, Johnson, and Vlissides, 1995]. 그림 8-3에 결과적인 설계를 보여주었다. 이 그림에서 추상클라스와 그 추상(가상)방법의 이름은 빗선체로 되여 있다(추상클라스는 비록 기초클라스로 리용될수 있다고 하여도 실체화될수 없는 클라스이다. 그것은 보통 적어도 하나의 방법을 포함하고 있다.). 그림 8-3의 꼭대기부분에 추상클라스 ***Abstract Widget Factory***가 있다(7.7에서 언급한바와 같이 UML규정에서는 클라스는 매 단어의 첫문자가 대문자인 굵은체로 되여 있어야 한다.). 이러한 추상클라스는 많은 추상방법들을 포함하고 있다. 간단히 하기 위해서 여기서는 다만 두가지 즉 create menu와 create window를 보여 준다. 그림에서 아래로 가면서 **Linux Widget Factory, Mac OS Widget Factory, Windows Widget Factory** 는 ***Abstract Widget Factory***의 구체적인 부분클라스들이다. 매 클라스는 주어 진 조작체계상에서 동작할수 있는 부분품을 창조하는 특정한 방법을 포함한다. 실례로 **Linux Widget Factory**에 있는 create menu는 차림표객체가 Linux상에서 동작할수 있도록 창조되게 한다.

또한 매개의 부분품에 대하여 추상클라스들이 있다. 여기서는 ***Abstract Menu***와 ***Abstract Window*** 두가지를 보여 준다. 매개가 다 세개의 조작체계에 대하여 각각 하나의 구체적인 부분클라스를 가지고 있다. 실례로 **Linux Menu**는 ***Abstract Menu***에 대한 하나의 구체적인 부분클라스이다. 구체적인 부분클라스 **Linux Widget Factory**안에 있는 방법 create menu는 **Linux Menu**형의 객체가 창조되게 한다.

창문을 창조하기 위하여 응용프로그람안에서 **Client**객체는 ***Abstract Widget Factory***의 방법 *create window*에 통보문을 보내야 하며 다형성은 정확한 부분품이 창조된다는것을 담보해 준다. 이제 응용프로그람이 Linux에서 동작한다고 하자. 우선 **Linux Widget Factory**형(클라스)의 객체 Widget Factory가 창조된다. 그다음 파라메터로서 Widget Factory를 넘겨 주는 ***Abstract Widget Factory***의 방법 *create window*를 추상화한 통보문은 구체적인 부분클라스 **Linux Widget Factory**안에 있는 방법 create window에 대한 통보문으로 해석된다. 차례로 방법 create window는 **Linux Window**를 창조하기 위해서 통보문을 보낸다. 즉 이것은 그림 8-3에서 제일 왼쪽에 있는 수직으로 그은 점선으로 표시하여 주고 있다.

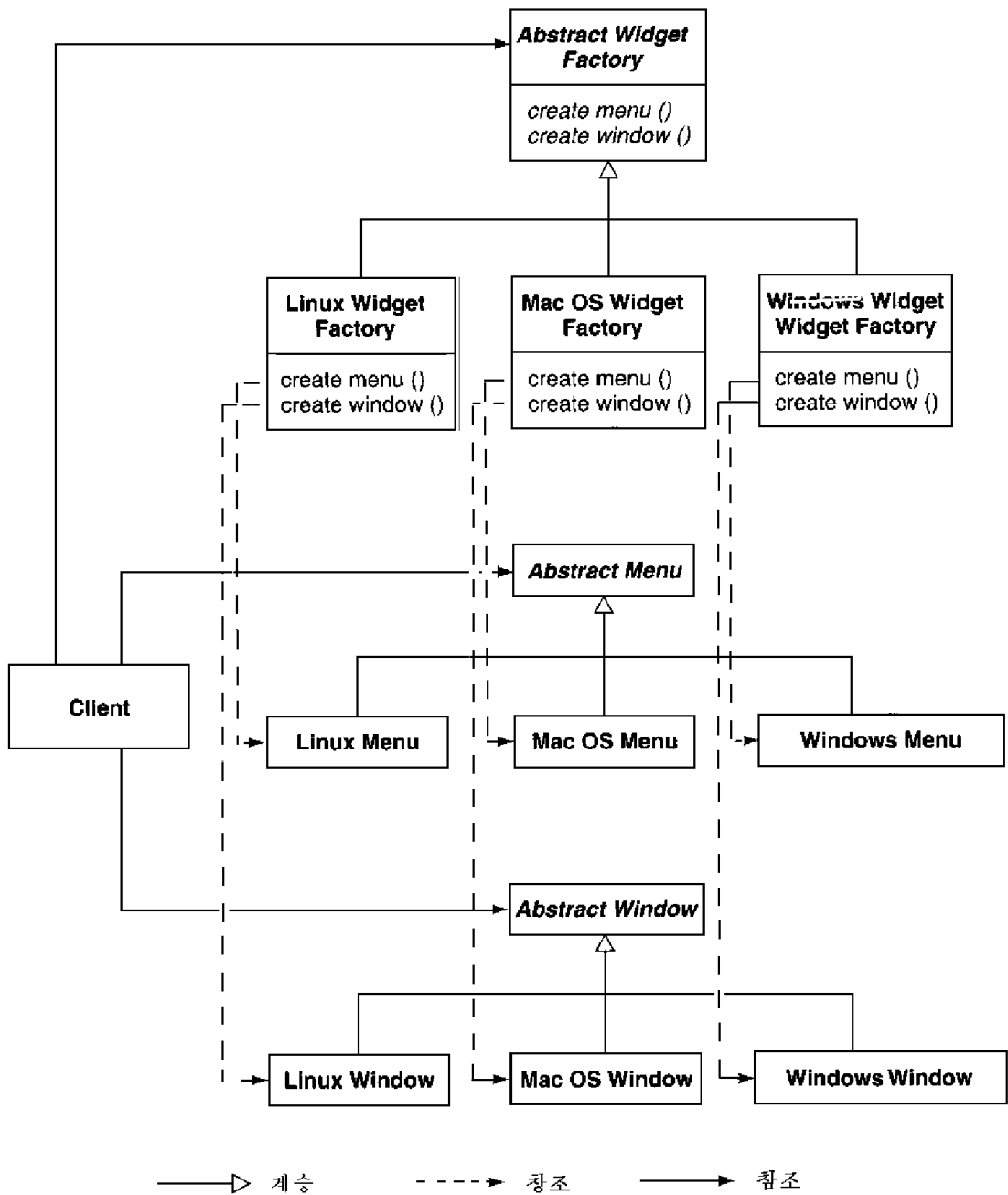



그림 8-3. 도형사용자대면부 도구들의 설계
추상클라스와 그 가상적인 함수들은 빗선체로 되여 있다.

이 그림에서 위험한 측면은 응용프로그람안에 있는 **Client**와 부분품생성프로그람클라스 ***Abstract Widget Factory, Abstract Menu, Abstract Window***들사이 세개의 대면부는 모두 추상클라스들이라는것이다. 추상클라스들의 방법들이 **abstract** (C++에서는 **virtual**)이

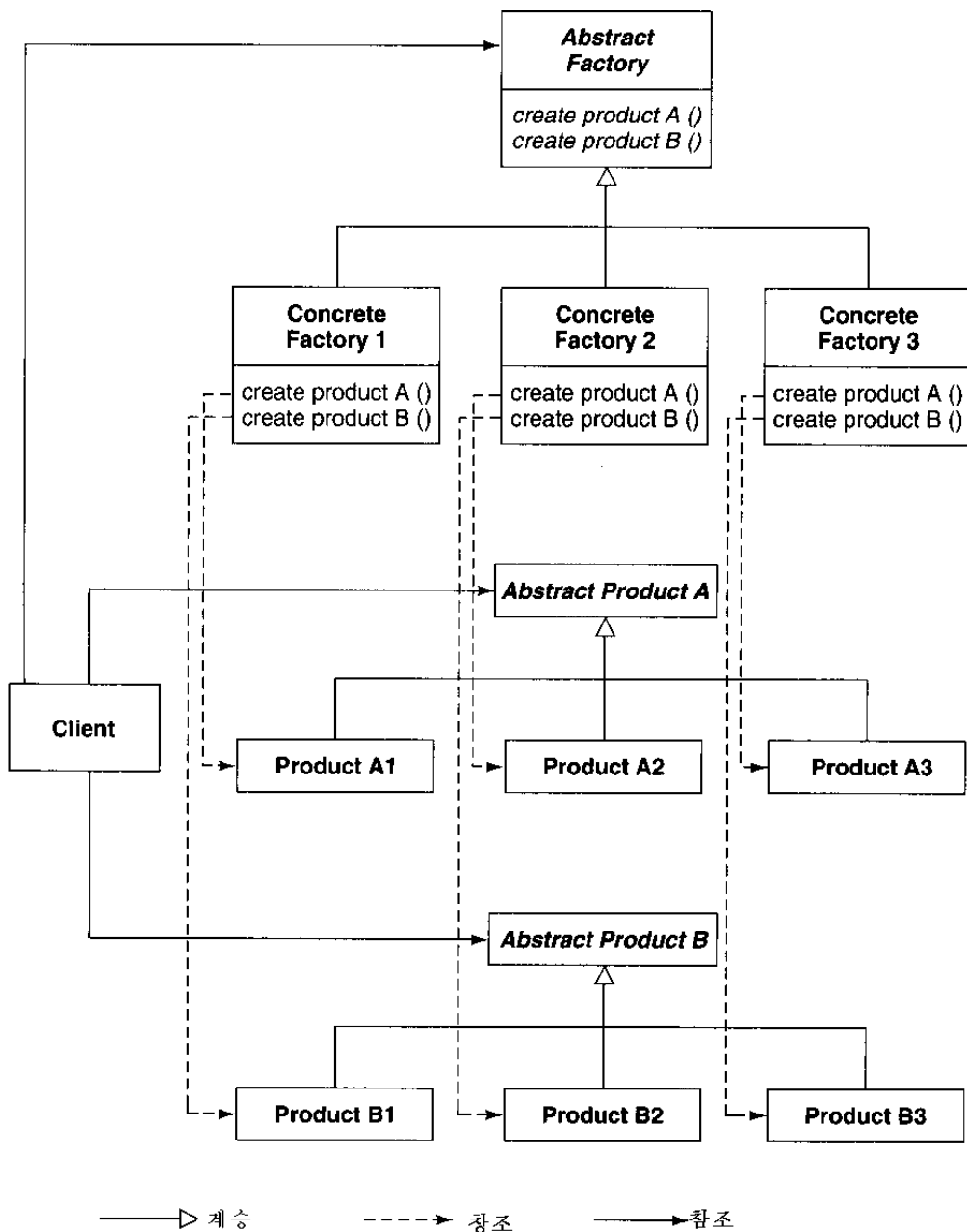



그림 8-4. 패턴 Abstract Factory
추상클라스와 그것의 가상함수들의 이름은 빗선체로 되여 있다.

기때문에 이러한 대면부들은 그 어떤 조작체계에 대하여 특정한것이 아니다. 이리하여 그림 8-3의 설계는 응용프로그람을 조작체계와 분리시켜 놓는다. 그림 8-3의 설계는 그림 8-4에서 보여 준 패턴 ***Abstract Factory***의 실체이다. 이 패턴을 리용하기 위하여 특정한 클라스들은 일반적인 이름들을 **Concrete Factory 2**와 **Product B3**과 같이 교체한다. 그때문에 그림 8-2의 ㄷ)에서는 설계패턴에 대한 기호적인 표현이 컴컴한 칸에 흰칸을 포함하고 있으며 흰칸은 설계에서 이러한 패턴들을 재리용하기 위하여 제공되여야 할 세부들을 보여 주고 있다. 패턴들은 다른 패턴들과 호상작용한다. 이것을 그림 8-2의 ㄹ)에서 기호적으로 표현해 주었다. 그림 8-2의 ㄷ)에서 중간에 있는 패턴들의 왼쪽아래 블로크도 패턴이다. 문헌 [Gamma, Helm, Johnson, and Vlissides, 1995]에 있는 문서편집기의 실례연구는 8개의 서로 다른 호상작용하는 패턴들을 포함하고 있다. 그것은 바로 실천에서 무엇이 일어 나는가 하는것이다. 즉 제품의 설계가 바로 하나의 패턴을 포함하고 있다는것은 흔히 있을수 있는 일이 아니다.

도구틀과 틀거리를 가지고 하기때문에 설계패턴들을 재리용하면 그 패턴의 실현도 역시 재리용할수 있다. 더우기 분석패턴도 객체지향분석을 방조해 줄수 있다[Coad, 1992; Fowler, 1997a]. 마지막으로 패턴외에도 반대패턴(*antipattern*)들이 있다. 즉 이 사실에 대하여서는 다음의 《알고 싶은 문제》에서 서술하고 있다.

알고 싶은 문제

반대패턴(*antipattern*)은 《분석마비》(분석단계에서 아주 많은 시간과 노력이 드는)나 하나의 객체가 거의 모든 작업을 한다고 하는 객체지향제품을 설계하는것과 같이 어떤 프로젝트를 실패시킬수 있는 행위이다. 처음에 반대패턴에 관한 책[Brown et al., 1998]을 쓰게 된 기본동기는 모든 쏘프트웨어제품들가운데서 거의 1/3이 취소되고 2/3는 비용이 200%로 초과되게 되고 80%이상은 실패라고 생각하였기때문이다.

8. 5. 4. 쏘프트웨어구성방식

쏘프트웨어제품의 구조는 객체지향적이며 관들과 려과기들(UNIX구성부분) 또는 의뢰자-봉사기로서 서술할수 있다. 그림 8-2의 ㄹ)은 도구틀과 틀거리 그리고 세개의 설계패턴들로 구성된 구성방식설계를 기호적으로 보여 주고 있다.

쏘프트웨어구성방식분야는 전체적인 제품의 설계에 응용되기때문에 그것의 구성부분에 의한 제품의 조직과 제품의 준위조종구조, 통신과 동기화문제, 자료기지와 자료호출, 구성부분들의 물리적인 분포, 성능, 설계방도의 선택을 포함하여 여러가지 설계상 문제점들을 내포하고 있다[Shaw and Garlan, 1996]. 이리하여 쏘프트웨어구성방식은 설계패턴보다도 훨씬 더 넓은 범위의 개념으로 된다.

사실상 쇼와 갈랜[Shaw and Garlan, 1996]은 다음과 같이 말했다. 《추상적으로 쏘프트웨어구성방식은 체계가 구축되는 요소들의 식별과 그리고 그러한 요소들속에서의 호상작용, **그러한 구성부분들을 안내해 주는 패턴들, 그러한 패턴들에 대한 제한**을 포함한다.》 앞의 단락에서 라렬한 많은 항목들을 제외하고도 쏘프트웨어구성방식은 부분적인 항목으

로서 패턴을 포함하게 된다. 이것은 바로 그림 8-2의 ㄷ)에서 쏘프트웨어구성방식의 구성요소들로서 세개의 설계패턴들을 보여 주고 있는가 하는 리유로 된다.

쏘프트웨어구성방식이 재리용될 때 설계의 재리용에 대한 우점들은 더욱더 커지게 된다. 구조의 재리용을 실천적으로 실현하기 위한 하나의 방도는 쏘프트웨어제품개발지침을 리용하는것이다[Lai, Weiss, and Parnas, 1999; Jazayeri, Ran, and van der Linden, 2000]. 그 착상은 바로 많은 쏘프트웨어제품에 공통인 쏘프트웨어구성방식을 개발하며 새 제품을 개발할 때 이 구조를 실증하는것이다. 휼레트-패카드회사는 아주 다양한 인쇄기들을 제작하였으며 새로운 모형을 끊임없이 개발하고 있다. 휼레트-패카드회사는 지금 매 새로운 인쇄기모형에 대하여 실증된 펌웨어구조를 가지고 있다. 그 결과는 매혹적이였다. 실례로 1995년과 1998년사이에 새로운 인쇄기모형에 대한 펌웨어를 개발하는데 드는 시간당 공수는 1/4로 줄어 들었고 그 펌웨어를 개발하는데 드는 시간은 1/3로 줄어 들었다. 그러나 재리용은 늘어 났다. 최근의 인쇄기들에서는 펌웨어구성부분의 70%이상이 재리용되고 있으며 원래의 제품들에서 거의나 변화되지 않았다[Toft, Coleman, and Ohta, 2000].

8. 6. 재리용과 유지정비

재리용을 촉진시키게 된 기본리유는 개발공정을 단축할수 있었다는데 있다. 실례로 많은 주요한 쏘프트웨어기업체들은 새 제품을 개발하는데 요구되는 시간을 절반으로 줄이려고 하고 있으며 재리용은 이러한 요구를 충족시키는데서 기본전략으로 되고 있다. 그러나 그림 1-2에서 보여 준바와 같이 제품을 개발하는데 1딸라가 소비되고 제품을 유지정비하는데는 2딸라가 소비된다. 그러므로 재리용을 하게 되는 두번째 리유는 제품을 유지정비하는데 드는 시간과 비용을 줄이려는데 있다. 사실상 재리용은 개발보다도 유지정비에 더 큰 영향을 주고 있다.

제품의 40%는 그보다 먼저 개발된 제품으로부터 재리용된 구성부분들로 구성되여 있으며 이러한 재리용은 전체 제품에 대하여 다같이 적용된다고 가정하자. 즉 명세서 40%는 재리용된 구성요소들로 구성되여 있으며 설계의 40%와 코드모듈의 40%, 지도서의 40% 등에 대하여서도 마찬가지로 고찰할수 있다. 그러나 이것은 전체적인 제품을 개발하는데 드는 시간이 재리용을 하지 않았을 때보다 40% 더 적다는것을 의미하지 않는다. 첫째로, 일부 구성부분들은 새 제품에 알맞게 고쳐 져야 한다. 재리용된 구성부분의 1/4이 변경된다고 가정하자. 구성부분이 변경되면 그 구성부분에 대한 문서도 변경된다. 이밖에 변경된 구성부분은 시험되여야 한다. 둘째로, 코드모듈이 변경되지 않고 재리용되면 그 모듈의 단위시험은 필요하지 않다. 그러나 그 모듈에 대한 통합시험은 여전히 필요하게 된다. 그래서 지어 제품의 30%가 변경되지 않고 재리용된 구성부분들로 구성되여 있다고 하면 완성된 제품을 개발하는데 필요되는 시간은 기껏해서 약 27% 더 적어 지게 된다[Schach, 1992]. 평균적으로 쏘프트웨어예산의 33%가 개발하는데 소비된다. 결국 재리용이 개발비용을 27%까지 줄이면 12년～15년까지 수명을 가진 제품의 전체적인 비용은 재리용의 결과로 대략 9%까지 줄어 들게 된다. 이 사실에 대해서는 그림 8-5에서 보여 주었다.

활동	제품수명에 대한 전체 비용의 퍼센트	재리용으로 인한 제품수명의 비용절약퍼센트
개발	33%	9.3%
유지정비	67	17.9

그림 8-5. 새 제품의 40%가 재리용된 요소로 구성되고 재리용된것의 3/4은 변화되지 않았다는 가정하에서 보여 준 평균 비용절약퍼센트

류사하면서도 보다 장황한 인수들은 쏘프트웨어개발공정의 유지정비구성부분에 응용할수 있다[Schach, 1994]. 앞에 진행한 가정대로 하면 유지정비에서 재리용의 효과는 그림 8-5에서 보여 준바와 같이 전반적으로 약 18%의 비용을 절약하는것으로 된다. 명백히 말해서 재리용이 주는 중요한 영향은 개발에 있는것이 아니라 유지정비에 있다. 기본리유는 바로 재리용된 구성부분들이 일반적으로 잘 설계되여 있고 철저하게 시험되였으며 리해하기 쉽게 문서화되여 있어 세가지 형태의 유지정비를 모두 간단히 진행할수 있게 한다는데 있다.

주어 진 제품에 대하여 실제적인 재리용률이 가정한것보다 더 낮으면(혹은 더 높으면) 재리용의 리익성은 달라 지게 된다. 그러나 전반적인 결과는 같다. 즉 재리용은 개발보다도 유지정비에 더 큰 영향을 미친다.

다음은 이식성에 대하여 보기로 한다.

8. 7. 이식성

쏘프트웨어의 비용이 끊임없이 늘어 나는것은 비용을 조절하는 수단을 찾아 낼것을 절박하게 요구하고 있다. 그 한가지 방도는 제품이 전체적으로 여러가지 종류의 하드웨어-조작체계결합에 대하여도 쉽게 동작할수 있도록 하는것이다. 제품을 개발하는데 드는 일부 비용은 다른 콤퓨터들에서도 동작할수 있도록 개발된 판본을 판매함으로써 보상될수 있다. 하지만 다른 콤퓨터들에서도 쉽게 동작할수 있는 쏘프트웨어를 개발하는 가장 중요한 리유는 4년에 한번씩 의뢰자측이 새로운 하드웨어를 구입하고 모든 쏘프트웨어들이 그 새로운 하드웨어에서 동작할수 있게 바꾸기때문이다. 새로운 제품을 령상태에서 개발하는것보다 그 제품을 새로운 콤퓨터에서도 동작할수 있게 하는것이 비용이 훨씬 더 적을 때 그 제품에 대하여 이식을 할수 있다[Mooney, 1990].

더 정확히 말하여 이식가능성은 다음과 같이 정의할수 있다. 어떤 제품 P가 콤파일러 C에 의하여 콤파일되여 원천콤퓨터상에서 실행된다고 하자. 이를테면 하드웨어구성배치 H가 조작체계 O에 놓인다. 기능상은 제품 P와 같지만 콤파일러 C'에 의하여 콤파일되고 목적콤퓨터에서 동작한다는데로부터 제품을 P`라고 한다. 즉 하드웨어구성배치 $H'$는 조작체계 $O'$에 놓인다. 제품 $P$를 제품 $P$`로 변환하는데 드는 비용이 제품 P'를 처음부터 새로 코드작성하는데 드는 비용보다 현저히 적다면 제품 P는 이식가능(*portable*)하다고 말한다.

전체적으로 볼 때 쏘프트웨어를 변환하는 문제는 각이한 하드웨어구성배치와 조작체계 그리고 콤파일러사이에 호환성이 없는것으로 하여 중요하게 제기된다. 이에 대하여 매 측면을 차례로 검토한다.

8. 7. 1. 하드웨어의 비호환성

현재 하드웨어구성배치 *H*에서 동작하는 제품 *P*를 하드웨어구성배치 *H'*에 설치하려고 한다. 피상적으로 보면 이것은 단순하다. 제품 *P*를 *H*의 하드구동기로부터 DAT테프에 복사하고 그것을 *H'*에 옮긴다. 하지만 만일 *H'*가 예비복사를 위하여 Zip구동기를 리용하였다면 이것은 옮길수 없다. 즉 DAT테프는 Zip테프구동기에서 읽을수 없다.

이제 제품 *P*의 원천코드를 콤퓨터 *H'*에 물리적으로 복사하는 문제가 해결되였다고 하자. 여기서 *H'*가 *H*에 의하여 만들어 진 비트패턴을 해석할수 있다는 담보는 없다. 일반적으로 여러가지 종류의 많은 문자코드가 있는데 여기서 가장 일반적인것은 확장2진식10진변환코드(EBCDIC)와 정보변환용표준코드(ASCII), 7bit ISO코드이다[Mackenzie, 1980]. 만일 *H*가 EBCDIC를 사용하고 *H'*가 ASCII방식을 리용한다고 하면 *H'*는 *P*를 쓸수 없는것으로 취급한다. 이와 류사하게 개인용콤퓨터 PC는 일반적으로 마킨토쉬형식의 자료를 읽을수 없으며 반대로 마킨토쉬는 PC형식으로 된 자료를 읽을수 있다.

이러한 차이가 있게 되는 력사적인 고찰이 있지만(즉 서로 다른 제작자들에 대하여 독립적으로 일하는 연구사들은 서로 다른 방법으로 같은 기능을 수행하는 제품을 개발하였다.) 그것들을 존속시키는데는 제한된 경제적리유가 있다. 이것을 고찰하기 위하여 다음과 같은 가상적인 환경을 생각해 보자. MCM콤퓨터제작업체는 수천대의 MCM-1 콤퓨터를 판매하였다. MCM은 현재 MCM-1보다 모든 면에서 성능이 더 높고 비용은 훨씬 더 적은 새 콤퓨터 MCM-2를 설계제작하여 팔려고 하고 있다. MCM-1이 ASCII코드를 리용하고 있으며 9bit짜리 4개로 구성된 36bit단어를 가지고 있다고 가정하자. 이제 MCM의 책임콤퓨터설계가는 MCM-2는 EBCDIC코드를 사용하며 8bit짜리 두개로 구성된 16bit단어를 가지고 있다고 하였다. 그다음 판매부서에서는 현제의 MCM-1소유자들에게 MCM-2이 그와 동일한 경쟁대상의 콤퓨터보다 3만 5천딸라 적게 비용이 들지만 현존쏘프트웨어와 자료를 MCM-1형식에서 MCM-2형식으로 변환하는데 20만딸라의 비용이 들게 될것이라고 말해 주어야 한다. MCM-2를 재설계하는 과학적인 론거가 아무리 훌륭하다고 해도 판매상 고려해야 할 문제는 새로운 콤퓨터가 낡은 콤퓨터와 호환가능하다는것을 담보해 주는것이다. 그다음 판매원은 현재의 MCM-1소유자들에게 MCM-2콤퓨터가 그 어떤 다른 경쟁자의 콤퓨터보다 비용이 3만 5천딸라 적게 들뿐아니라 잘못 알고 다른 제작자에게서 산 손님들은 3만 5천딸라로 지내 비용을 많이 들였다는것과 또한 현존쏘프트웨어와 자료를 비MCM콤퓨터형식으로 전환시키는데 비용이 약 20만딸라를 들여야 한다고 말해 줄수 있다.

앞에서 본 가상적인 환경에서 실지 현실세계로 돌아 와서 보면 현재까지 가장 성공적인 계렬의 콤퓨터는 IBM system/360-370계렬이였다[Gifford And Spector, 1987]. 이러한 계렬의 콤퓨터들이 성공하게 된것은 바로 그러한 콤퓨터들사이에 완전한 호환성이 있는데 있다. 즉 1964년에 개발된 IBM System/360 모형 30상에서 동작하는 제품은 2001년에 개발된

IBMS/390모형 IC5상에서 변화시키지 않고도 동작할수 있다. 그러나 IBM System/360 모형 30상의 OS/360체계하에서 동작하는 제품은 Sun Enterprise 10000상의 Solaris체계와 같이 전혀 다른 2001년 기계상에서 동작시키기전에 상당한 수정을 진행하여야 한다. 난관의 일부는 바로 하드웨어의 비호환성과 조작체계의 비호환성때문이다.

8. 7. 2. 조작체계의 비호환성

임의의 콤퓨터에 대하여 일감조종언어(JCL)는 일반적으로 아주 다르다. 이러한 일부 차이는 문장론적이다. 즉 수행적재가능한 프로그람을 실행시키기 위한 지령은 한 콤퓨터에서는 @xeq, 다른 콤퓨터에서는 //xqrt, 세번째 콤퓨터에서는 .exc일수 있다. 제품을 다른 조작체계상태로 바꿀 때 문장론적차이는 하나의 JCL로부터 다른 JCL로 지령을 번역하여 간단히 조절할수 있다. 그러나 기타 다른 차이는 더 심각하다. 실례로 일부 조작체계는 가상기억을 지원하고 있다. 이제 조작체계가 제품에 대하여 크기상 128Mb까지 허락하지만 특정한 제품에 대하여 할당된 실제의 주기억령역은 8Mb라고 하자. 그러면 사용자의 제품이 256Kb크기의 페지들로 분할배치되고 다만 이 페지들가운데서 32개의 페지가 일정한 시간 기억에 상주하게 된다. 나머지 페지들은 디스크상에 있으며 가상기억조종체계에서 요구한대로 교환이 이루어 진다. 결과 제품은 크기상 효과적인 제한을 받지 않고 작성될수 있다. 그러나 가상기억조종체계하에서 성과적으로 실현되는 제품이 제품크기상 물리적인 제한이 있는 조작체계상에서 동작하게 되자면 전체 제품은 크기상 제한을 받지 않도록 겹쳐놓이기법을 리용하여 다시 작성하여 련결하여야 한다.

8. 7. 3. 수자처리쏘프트웨어의 비호환성

제품이 한 기계로부터 다른 기계로 이식될 때 혹은 지어 서로 다른 콤파일러에 의하여 콤파일될 때 산수연산처리결과는 서로 다를수 있다. 16bit기계 즉 단어크기를 16bit로 하고 있는 콤퓨터상에서 옹근수는 보통 한 단어(16bit)로 표현되며 두배정확도옹근수는 두개의 련결된 단어(32bit)로 표현된다. 그러나 일부 언어실현에서는 두배정확도옹근수를 포함하지 않고 있다. 실례로 표준적인 Pascal은 두배정확도옹근수를 포함하지 않고 있다. 그러므로 옹근수들이 오직 16bit로 표현되는 콤퓨터로 이식될 때 Pascal에서 옹근수가 32bit를 리용하여 표현되는 콤파일러와 하드웨어-조작체계구성배치상에서 정확히 기능을 수행하는 제품들은 실패할수도 있다. 2^{16}보다 더 큰 옹근수를 류동소수점수(형태는 실수)로 표현하는것은 명백히 할수 없다. 왜냐하면 옹근수는 정확히 표현할수 있지만 류동소수점수는 일반적으로 오직 가수부와 지수부를 리용하여 근사적으로 표현할수 있기때문이다.

Ada는 옹근수형의 범위와 류동소수점수형태의 정확도를 설정할수 있기때문에 Ada에서는 이 문제를 해결할수 있다. Ada–유럽이식성처리그룹은 Ada의 이식성을 담보하기 위한 앞으로의 권고목록을 작성하였다[Nissen and Wallis, 1984].

Java와 관련하여 8개의 기본자료형이 설정되였다. 실례로 형 **int**는 항상 부호 있는 32bit 두개의 보수로 실현하였고 형 **float**는 항상 32bit를 차지하고 류동소수점수를 위한 IEEE규격754를 만족시킨다[ANSI/IEEE 754, 1985]. 따라서 Java에서는 수계산을 목적으로

하는 하드웨어-조작체계상에서 정확히 진행하도록 하는 문제는 제기되지 않는다(Java의 설계에 대하여 좀 더 알고싶으면 다음의 《알고 싶은 문제》를 보시오). 그러나 수계산을 Ada나 Java와 다른 언어에서 진행하는것은 중요하지만 수계산을 목적하는 하드웨어-조작체계상에서 정확히 진행하도록 하는것은 때때로 곤난하다.

알고 싶은 문제

1991년에 싼 마이크로시스템(Sun Microsystems) 제임스 고스링(James Gosling)은 Java를 개발하였다. 그 언어를 개발하는동안 그는 자주 창문너머로 사무실밖에 있는 큰 참나무를 살펴 보군 하였다. 그는 이런 행동을 너무 자주 하여서 새로 만든 언어를 참나무라고 이름 짓기로 하였다. 그러나 싼회사는 그가 선택한 이름을 허락하지 않았다. 왜냐하면 그것이 회사의 등록상표가 아니였고 등록상표가 없이는 싼회사가 언어에 대한 관리를 할수 없기때문이였다.

등록상표를 달고 상기하기 쉬운 이름을 찾기 위하여 노력한 결과 고스링그룹은 Java로 하게 되였다. 18세기에 영국에 들어 온 많은 커피들은 더취 동인디아(Dutch East India)(지금의 인도네시아)에 있는 가장 인구가 조밀한 섬인 Java에서 자란것들이였다. 결국 Java는 커피를 가리키는 전문단어로 되였고 쏘프트웨어공학자들에게는 세번째로 가장 인기있는 청량음료였다. 유감스럽게도 탄산가스화한 콜라음료의 이름은 이미 등록상표를 달고 있었다.

왜 고스링이 Java를 설계하였는가를 리해하기 위하여서는 그가 C++에서 깨닫게 된 약점의 근원을 정확히 알아야 한다. 그래서 C++의 조상언어인 C로 되돌아 가보자.

1972년에 프로그람작성언어 C는 AT&T벨연구소에 있는 데니스 리치에(Dennis Ritchie)가 체계쏘프트웨어에서 리용하기 위하여 개발하였다. 언어는 아주 융통성 있게 설계되였다. 실례로 언어는 지적자변수로 산수연산을 진행하도록 하고 있다. 실례로 C언어는 지적자변수 즉 기억주소를 보관하는데 리용하는 변수를 가지고 산수연산을 진행하도록 하고 있다. 보통 프로그람작성자의 관점에서 보면 이것은 명백히 위험성을 내포하고 있다. 결과 프로그람은 조종이 콤퓨터의 임의의 곳으로 넘어 가기때문에 아주 불안할수 있다. 또한 C는 배렬을 구체적으로 표현하지 못한다. 대신에 배렬의 첫 시작을 가리키는 주소에 대한 지적자가 리용된다. 결과 배렬범위가 초과되는 개념은 C에 대하여 본질적인것이 아니다. 이것은 불안정성의 근원으로 된다.

이러한 불안정성은 벨연구소에서는 문제가 없었다. 결국 C언어는 벨연구소에 있는 다른 경험 있는 쏘프트웨어공학자들이 리용하기 위하여 경험 있는 쏘프트웨어공학자들이 설계하였다. 전문가들은 C언어의 성능이 높고 융통성 있는 특성이 안전하게 리용되리라고 믿었다. C언어의 설계에서 기본관념으로 삼은것은 C언어를 리용하는 사람들이 자기가 무엇을 하고 있는가를 정확히 알도록 하는것이였다. 능력이 부족하거나 경험이 없는 프로그람작성자들이 C언어를 리용할 때 초래될수 있는 쏘프트웨어실패는 AT&T에게는 책임이 없었다. 즉 C언어는 오늘과 같이 일반목적의 프로그람작성언어를 널리 쓰이도록 하여야 한다는 의도는 전혀 없었다.

객체지향파라다임이 출현하여 ObjectC, ObjectiveC, C++를 비롯한 많은 객체지향프로그람작성언어들이 C언어에 기초하여 개발되였다. 이러한 언어들의 리면에 있는 착상은 당시로서는 인기 있는 프로그람작성언어였던 C언어에 객체지향적인 구축물을 끼워 넣은

것이였다. 완전히 새로운 문법을 배우는것보다도 잘 알려 진 언어에 기초한 언어를 배우는것이 프로그람작성자에게 있어서 더 쉬울것이라는 주장도 있었다. 그러나 다만 많은 C언어에 기초한 객체지향언어들가운데서 하나의 언어 C++만이 널리 쓰이게 되였는데 C++언어는 역시 AT&T벨연구소의 브쟈네 스트로우스트루프(Bjarne Stroustrup)가 개발하였다.

C++언어의 성공의 비결이 AT&T의 막대한 재정적인 힘에 있다고 보는 견해도 있다. 그러나 만일 프로그람작성언어를 설정하는것이 회사의 크기와 재정적인 힘과 관련된다고 하는 특징이 있다고 한다면 오늘 모두가 IBM이 개발하여 요란하게 선전한 PL/I를 사용하고 있을것이다. IBM의 PL/I이 밀려 난것은 현실이다.

C++언어가 성공한 실지의 리유는 그것이 실지 C언어를 초월한 언어라는데 있다. 즉 그 어떤 다른 C언어에 기초한 객체지향프로그람작성언어들과는 달리 실제상 임의의 C언어도 타당한 C++언어인것이다. 따라서 기업체들은 자기들의 현존 C쏘프트웨어를 교체하지 않고도 C언어에서 C++언어에로 전환할수 있다는것을 깨달았다. 그들은 구조화파라다임으로부터 객체지향파라다임으로 끊임없이 전진할수 있었다. Java에 대한 문헌들에서 자주 맞다들게 되는 말은 《Java는 C++가 되였어야 했을것이다.》이다. 그밖에 스트로우스트루프가 고스링처럼 총명하기만 하였더라면 C++언어가 Java언어로 되였을것이라는것이다. 반대로 만일 C++언어가 진짜 C언어를 초월한 언어로 되지 못했더라면 다른 모든 C언어에 기초한 객체지향프로그람작성언어들과 같은 길을 걸었을것이다. 즉 불필요 사라졌을것이다. C++언어가 인기 있는 언어로서의 지위를 차지한후에야 C++언어에 있는 약점에 반응하여 Java언어가 설계되였다. Java언어는 C언어를 초월한 언어가 아니다. 실례로 Java언어에는 지적자변수가 없다. 그러므로 《Java언어는 C++언어가 도저히 될수 없었던 것이다.》라고 말하는것은 보다 정확한것이다.

끝으로 Java언어가 다른 모든 프로그람작성언어들처럼 자체의 약점을 가지고 있다는 것을 알아야 한다. 그밖에도 일부 령역(호출규칙과 같은)에서 C++언어는 Java언어보다 우월하다[Schach, 1997]. 앞으로 C++언어가 계속 지배적인 객체지향프로그람작성언어로 되겠는지 Java언어나 어떤 다른 언어가 그 자리를 대신 차지하겠는지 하는것은 두고 보아야 할것이다.

8. 7. 4. 콤파일러의 비호환성

이식성은 어떤 제품이 그에 대한 콤파일러가 거의나 존재하지 않는 언어로 실현된다면 이루어 질수 없다. 만일 그 제품이 CLU[Liskov, Snyder, Atkinson, and Schaffert, 1977]와 같은 전문언어로 실현된다면 목적콤퓨터가 그 언어를 위한 콤파일러를 가지고 있지 못한 경우에 그것을 다른 언어로 다시 작성하여야 한다. 다른 한편 어떤 제품이 COBOL, FORTRAN, Lisp, Pascal, C, C++나 Java와 같은 일반언어로 실현된다면 목적하는 콤퓨터를 위한 그런 언어의 콤파일러나 해석프로그람을 찾을수 있다.

어떤 제품이 표준 FORTRAN과 같은 일반적인 고급언어로 작성되였다고 하자. 리론상으로는 그 제품을 한 콤퓨터에서 다른 콤퓨터로 이식하는데 문제가 없다. 그러나 사실은 그렇지 않다. Fortran 95[ISO/IEC 1539-1, 1997]로 알려 진 ISO/IEC FORTRAN 규격이 있기는 하지만 콤파일러 작성자가 거기에 매달릴 근거는 없다(Fortran 95에 대하여 더 알고 싶으면 다음의 《알고 싶은 문제》를 보시오.). 실례로 판매부서가 《새로운 확장된 FORTRAN 콤파일러》를 권유할수 있도록 FORTRAN에서 흔히 찾아 보지 못하는 추가적인 특징을

지원할 결정을 할수 있다. 반대로 극소형콤퓨터의 콤파일러가 완전한 FORTRAN실현을 할수 없을수도 있다. 또한 콤파일러를 개발하는데 기한이 주어 졌다면 관리자는 후에 수정하여 완전한 표준형을 지원할 의도에서 미완성품을 내놓기로 결정할수도 있다. 원천콤퓨터에 있는 콤파일러가 Fortran 95의 초월언어를 지원한다고 가정하자. 또한 목적콤퓨터가 표준Fortran 95의 실현이라고 가정하자. 그 원천콤퓨터에서 실현된 제품이 목적콤퓨터에 이식될 때 초월언어로부터 구성된 비표준Fortran 95를 리용하는 제품의 부분을 다시 코드작성하여야 한다. 그러므로 이식성을 보장하기 위하여 프로그람작성자들은 표준 FORTRAN언어의 특징만을 리용하여야 한다.

알고 싶은 문제

프로그람작성언어의 이름은 그것이 준말일 때 큰 글자로 쓴다. 실례로 ALGOL ALGOrithmic Language), COBOL(COmmon Business Oriented Language)과 FORTRAN(FORmula TRANSlator)을 들수 있다. 반대로 기타 다른 모든 프로그람작성언어들은 큰 글자. 시작하며 이름에 있는 나머지글자(있다면)들은 작은 글자로 되여 있다. 실례로 Ada, C C++, Java와 Pascal을 들수 있다. 그런데 FORTRAN규격위원회는 1990년판부터 시작하여 Fortran으로 쓰기로 결정하였다.

초기 COBOL규격은 미국의 콤퓨터제조업체들과 정부 및 개인사용자위원회인 Conference on DAta SYstems Language(CODASYL)가 개발하였다. 국제규격화기구(ISO), 국제전자기술위원회(IEC)의 제22차소위원회의 제1공동기술위원회가 현재 COBOL규격을 책임지고 있다[Schricker, 2000]. 그런데 COBOL규격은 이식성을 조장하지 않았다. COBOL규격은 공식적으로는 수명이 5년이지만 매개 성공적인 규격이 결코 그 이전의것에 대한 초월언어인것은 아니다. 또한 우려되는것은 많은 특징이 개별적인 실현자들에게 맡겨지며 부분언어(하위언어)는 표준COBOL이라는 용어로 불리우며 그 언어를 확장하여 초월언어를 구성하는데는 제한이 없다[Wallis, 1982].

현재의 COBOL의 초보적인 표준언어인 OO-COBOL(2002년에 최종승인을 받을것으로 계획됨)은 완전히 객체지향적이다[ISO/IEC 1989, 2000]. 대비적으로 Fortran 95는 단지 객체에 기초한것이며[Wegner, 1992] 즉 계승과 클라스는 Fortran 95에서는 실현되지 않는다. Fortran 95 객체들은 따라서 그자체의 권한에 있는 실체이며 클라스의 실체는 아니다. 그 다음의 FORTRAN규격인 Fortran 2000은 완전히 객체지향적인것으로 볼수 있는데 이 책을 쓸 당시 Fortran 2000규격의 발표예정날자는 2002년 11월이다.

몇개의 서로 다른 Pascal규격들이 있다. 처음으로 젠센(Jensen)과 워스(Wirth)가 그 언어에 대한 정의를 하였고[Jensen and Wirth, 1975] 그다음은 ANSI규격(ANSI/IEEE 770X3.97, 1983)였다. 이렇게 규격이 지나치게 많음에도 불구하고 Pascal의 부분언어와 초월언어는 풍부하다. 실례로 모든 Pascal규격은 처리절차와 함수이름들이 인수로서 넘겨 질수 있다고 명기되여 있다. 하지만 그 특징이 결코 Pascal콤파일러가 일반적으로 지원하는것은 아니다. 반대로 Pascal의 많은 초월언어들이 실현되였다. 실례로 Pascal의 일부 실현은 모름지기 C언어와 경쟁하기 위하여 비트별 **and**와 **or**와 같은 비표준비트처리조작을 병합하고

있다. 또한 현재 수많은 Pascal실현은 Pascal에 대한 객체지향확장을 포함하고 있다.

미국국가규격협회(ANSI)는 프로그람작성에서 C에 대한 규격을 승인하였다[ANSIX3.159, 1989]. 그 규격도 1990년에 ISO가 승인하였다. 대부분의 C콤파일러들은 원래언어의 규격을 매우 엄격하게 고수하고 있다[Kernighan and Ritchie, 1978]. 이것은 거의 모든 C콤파일러 작성자들이 이식성이 있는 C콤파일러 *pcc*[Johnson, 1979]의 표준앞단을 리용하기때문이다. 결과 대다수의 콤파일러들에 의해서 인정되는 언어들은 동일하게 된다. 일반적으로 C언어제품은 하나의 실현에서 다른 실현에 쉽게 이식된다. C언어의 이식성을 보조하는 것은 **lint**처리기이다. 그것은 제품이 목적콤퓨터에 이식될 때 난관을 초래할수 있는 구성뿐아니라 실현의존적인 특징을 결정하는데 사용할수 있다. 유감스럽게도 **lint**는 문장론과 정적인 의미론만을 검사하며 따라서 그릇되게 증명하지 않는다. 하지만 앞으로의 문제들을 해결하는데서 상당한 도움이 될수 있다. 실례로 C언어에서 옹근수값을 지적자에 할당하는것이 합법적이며 또 지적자를 옹근수값에 할당하는것도 합법적이다. 하지만 **lint**에 의해서는 금지된다. 일부 실현에서 옹근수와 지적자의 크기(비트수)는 같은것이지만 크기들이 다른 실현에서는 다를수 있다. 이전종류의 잠재적인 미래의 이식성문제는 **lint**에 의하여 정지될수 있으며 거슬리는 부분은 다시 코드작성함으로써 제거될수 있다.

C++언어의 규격(ISO/IEC 14882, 1998)은 1997년 11월에 여러 국가규격위원회(ANSI포함)의 한결같은 찬동을 받았다. 그 규격은 1998년에 최종비준을 받았다. 지금까지 유일하게 성공적인 언어규격은 Ada참고지도서 [ANSI/MIL-STD-1815A, 1983]에서 구체화된 Ada83규격이다(Ada에 대한 배경지식을 알고 싶으면 다음의 《알고 싶은 문제》를 보시오.). 1987년말까지 Ada라는 이름은 Ada공동프로그람사무소(AJPO)의 등록상표였다. 상표의 소유자로서 AJPO는 Ada라는 이름이 규격과 정확히 일치하는 언어실현들을 위해서만 법적으로 리용될수 있다고 제정하였다. 그러나 부분언어와 초월언어는 특히 금지되였다. Ada콤파일러들을 타당하게 하기 위한 기구가 설치되였으며 타당성검사공정에 성과적으로 통과된 콤파일러만이 Ada콤파일러라고 불리울수 있었다. 그리하여 등록상표는 규격과 그로부터 이식성을 시행하는 의미로 리용되였다.

Ada라는 이름이 더는 등록상표가 아니기때문에 규격의 시행은 다른 기구들을 통하여 수행되고 있다. 타당성검사를 받지 못한 Ada콤파일러는 시장에 거의 없거나 전혀 없다. 그러므로 강한 경쟁력은 Ada콤파일러개발자들이 타당성검사를 받고 그로부터 Ada규격과 일치하는것으로 확인된 자기들의 콤파일러를 가지도록 하고 있다. 이것은 Ada83[ANSI/MIL-STD-1815A, 1983]과 Ada95 [ISO/IEC 8652, 1995]를 위한 콤파일러들에 적용하였다.

알고 싶은 문제

1970년대 초에 미국방성(DoD)은 자기의 쏘프트웨어와 관련한 문제들을 예민하게 인식하게 되였다. 보다 걱정스러운 문제의 하나는 적어도 450개의 서로 다른 언어가 DoD제품들에 사용되고 있다는 사실이였다. 이런 확산에 포함된 주요한 의미는 보통 환경에서도 어려운 유지정비가 이런 언어의 혼란을 해결할 능력이 있는 유지정비프로그람작성자들을 찾아야 하므로 거의 불가능한것으로 되였다는것이다. 이와 함께 쏘프트웨어개발 및 유지정비를 지원할 도구가 1차적으로 나서는데 주로는 그 매 언어를 위한 충분한 CASE도구들을 사거나 구축하는데 막대한 비용이 들기때문이였다.

내장형쏘프트웨어(내장형콤퓨터에 있는 쏘프트웨어, 6.4.4을 볼것)와 관련하여 형편은 매우 한심하였다. 매개 무력군종들이 자기 마음에 드는 실시간언어를 가지고 있었다. 즉 륙군은 TACPOL을 지원하고 있고 해군은 CMS-2를 장려하였으며 공군이 선택한것은 JOVIAL이였다. 간단히 말하여 내장형쏘프트웨어와 관련된 형편은 불리하였고 유지 정비에 대한 내용을 명심한다면 더 불리해 질수밖에 없었다.

내장형DoD쏘프트웨어는 수십, 수백만의 코드행으로 커지게 되고 10~15년의 수명을 가지고 있으며 그 기간에 요구가 변화되는데 따라 자주 변화되게 되는 경향이 있었다[Fisher, 1976]. 더우기 내장형쏘프트웨어는 거의나 항상 땅크, 무인조종기나 직승기와 같은 군사장비안에 내장된 콤퓨터의 크기가 일반적으로 제한적이라는 점에서 볼 때 공간상의 제약을 받기 쉽다. 더구나 실시간쏘프트웨어의 시간제한이 항상 존재한다. 마지막으로 내장형쏘프트웨어가 고도로 믿음성이 있는것이여야 한다. 쉽게 말하면 일단 탄도미사일이 핵잠수함에서 발사되였다면 그 어떤 종류의 실패가 발견되는 경우에 시간적으로 때늦으므로 그것에 맞게 쏘프트웨어를 수정할수 없게 된다.

1975년에 DoD는 모든 무력군종들에 일반화할수 있는 고급언어를 찾아 내기 위한 세계적인 경쟁을 발기하였다. 대학생, 산업전문가, 군사전문가들이 포함되여 있는 팀들로부터 17건의 제기를 받았는데 그중 4건은 더 발전시키기로 하였다. 심사자들이 경쟁기업체들의 신분을 모르게 하기 위하여 제안된 언어들에 Blue, Green, Red, Yellow라는 코드이름을 달아 주었다. 총체적으로 우승자는 프랑스의 호네이웰 불(Honeywell Bull)의 젠 이츠바흐(Jean Ichbiah)가 이끄는 주로 유럽팀인 Green이 되였다. 그래서 Ada참고지도서[ANSI/MIL-STD-1815A, 1983]에 풀색표지를 하게 되였다. 최근까지 계산기계협회의 Ada에 관한 특별리권그룹(ACM SIGAda)에 의하여 출판된 주요 Ada잡지인 *Ada Letters*들도 풀색표지를 하였다. 그러나 DoD는 초기의 이름 DoD-1처럼 Green이라는 이름이 마음에 들지 않았다. 해군의 잭크 쿠퍼(Jack Cooper)가 시인 로드 바이론(Lord Byron)의 딸인 라브레이스(Lovelace)의 백작부인 Ada의 이름을 따서 이름을 Ada라고 할것을 제기하였다. 그 녀자는 19세기전반기에 첫 콤퓨터인 Babbag의 분석엔진을 위한 프로그람을 작성하였다. Babbag의 설계는 정확하였지만 19세기 기술의 제한성으로 하여 분석엔진은 제작할수 없었다. 국방성차관은 라브레이스의 후계자인 리톤(Lytton)백작의 부인으로부터 Ada라는 이름을 사용할것을 허락 받았다[Carlson, Druffel, Fisher, and Whitaker, 1980]. 바로 그래서 언어의 이름을 ADA가 아니라 Ada로 쓰게 되였다. 그것은 략어가 아니다. 그것은 세계의 첫 프로그람작성자인 라브레이스백작의 부인인 Ada의 이름을 딴것이다.

언어는 DoD의 승인을 받아 군사규격 MIL-STD-1815으로 결정되였다. 수자 1815는 라브레이스의 출생년도이다. 1994년에 Ada의 수정판이 국제규격화기구와 국제전자기술위원회의 승인을 받았다. 그 규격 [ISO/IEC 8652, 1995]이 1995년에 출판되였으므로 언어를 Ada 95라고 부른다. 여러가지 새로운 특징들 특히는 객체지향성이 원판(Ada 83)에 첨가되였다.

1997년에 국가과학원 국가연구리사회의 충고에 따라 DoD는 Ada가 자기의 모든 쏘프트웨어들에 사용된다는 규정을 버리고 Ada를 다만 협소한 의미에서 쓰이는 술어인 《전후》쏘프트웨어로만 요구하였다[AdaIC, 1997]. Ada의 권한에서 일어 난 이러한 변화를 보아 Ada의 미래는 이전처럼 밝지 못하다.

타당성검사증명서는 특정한 조작체계와 특정한 하드웨어에서 실행되는 특정한 콤파일러를 위한것이다. 만일 Ada콤파일러를 개발한 기업체가 콤파일러를 다른 하드웨어나 조작체계구성배치에 넘길것을 바란다면 다시 타당성검사를 해야 한다. 모든 Ada콤파일러를 위한 참고지도서는 부록 6을 포함하고 있는데 거기에는 Ada실현에 관한 실현의존적인 특성이 서술되여 있다. 실례로 형변환을 취소하고 인수를 단순히 비트패턴으로 취급할수 있다. 그런데 실현은 그러한 항목의 크기에 대하여 제한을 가할수 있다.

그러면 기술적으로 Ada제품들을 부록 6에서 언급된 특성들과 관련한것을 제외하고는 완전히 이식가능하게 만들수 있다.

Java가 완전히 이식가능한 언어로 되자면 언어를 규격화하고 그 규격을 엄격히 준수하도록 하는것이 필수적이다. Ada공동프로그람사무소와 같이 Sun Microsystems은 규격화실현을 위하여 법적체계를 리용하고 있다. Sun은 저작권을 얻을수 있는 새로운 언어를 위한 이름을 선택하였다. Sun은 자기들의 저작권을 실시하는 이른바 위반자들에 대하여 법적행동을 취할것으로 보고 있다. 결국 이식성은 Java의 가장 위력한 특성들중의 하나이다. 만일 Java의 많은 판본들이 허용된다면 Java의 이식성은 진통을 겪게 될것이다. 즉 Java는 유독 모든 Java프로그람들이 모든 Java콤파일러에 의하여 똑같이 처리될 때에만 실지 이식가능하게 될수 있다. 사회여론에 영향을 주기 위하여 1997년에 Sun은 《순수한 Java》라고 선전깜빠니아를 벌렸다.

Java의 판본 1.0은 1997년 초에 나왔다. 론평들과 비평들에 대응하여 여러가지 개정된 판본들이 뒤따라 나왔다. Java의 이러한 단계적인 세련과정은 계속될것이다. 언어가 점차적으로 수정되자면 ANSI나 ISO와 같은 규격화기구가 초안규격을 공개발표하고 세계 각지로부터 론평을 받아야 한다. 이러한 론평들은 공식적인 Java규격을 결속하는데 리용될것이다.

8. 8. 왜 이식성이 필요한가

쏘프트웨어를 이식하는데서 나서는 많은 난관들과 관련하여 독자들은 결국 쏘프트웨어를 이식할 가치가 있는가 하는데 대하여 의심할수 있다. 8.7에서 언급된 이식성이 유익하다는 론거는 쏘프트웨어의 비용이 아마도 여러가지 하드웨어-조작체계구성배치에 대하여 제품을 이식함으로써 부분적으로 공제될수 있었다는것이다. 그러나 쏘프트웨어의 다양한 변종들을 판매하는것은 불가능하다. 응용프로그람은 고도로 전문화될수 있고 그 어떤 다른 의뢰자도 쏘프트웨어를 필요로 하지 않을수도 있다. 실례로 어느 한 주요 승용차임대회사를 위하여 개발된 관리정보체계는 다른 승용차임대회사들의 운영에 적용할수 없을수도 있다. 선택적으로 쏘프트웨어 그자체는 의뢰자에게 경쟁할수 있는 유리성을 제공해 줄수 있으며 제품을 복사하여 판매하는것은 경제적으로 자살행위와 다름이 없게 될것이다. 이 모든 사실들에 비추어 볼 때 이식을 제품이 설계되였을 때 공학적인 방법으로 제품화하는것이 시간과 비용의 랑비이겠는가?

이 물음에 대한 대답은 강조해서 말하여 아니다라는것이다. 이식이 필수적인것으로 되는 주요한 리유는 쏘프트웨어제품의 수명이 일반적으로 그것이 처음 개발된 하드웨어

의 수명보다 더 길다는것이다. 좋은 쏘프트웨어제품은 15년 혹은 그이상의 수명을 가질 수 있다. 반면에 하드웨어는 번번히 최소한 4～5년마다 변화된다. 따라서 좋은 쏘프트웨어는 자기의 수명에 비하여 3개 혹은 그이상의 각이한 하드웨어구성배치들에서 실현될수 있다.

이 문제를 해결하기 위한 한가지 방도는 다음과 같다. 방도는 상승식으로 호환가능한 하드웨어를 구입하는것이다. 거기서 유일한 비용은 하드웨어의 원가뿐이다. 쏘프트웨어는 변화시킬 필요가 없을것이다. 그럼에도 불구하고 일부 경우에 제품을 각이한 하드웨어에 완전히 이식하는것은 경제적으로 보다 안정할수 있다. 실례로 제품이 첫번째 판본을 7년전에 대형콤퓨터상에서 실현될수 있었다고 하자. 비록 그 제품이 그 어떤 변화도 없이 실행될수 있는 새로운 대형콤퓨터를 구입하는것이 가능할수 있어도 매개 사용자들이 탁상우에 하나씩 놓는 개인용콤퓨터망우에서 제품의 다양한 복사본들을 실현시키는것은 아주 덜 비싼것 같다. 이 실례에서 만일 쏘프트웨어가 이식을 도모하는 방식으로 개발되였다면 제품을 개인용콤퓨터망우에서 이식하는것은 재정적으로도 좋다.

하지만 다른 종류의 쏘프트웨어들이 있다. 실례로 개인용콤퓨터를 위한 쏘프트웨어를 개발하는 많은 기업체들은 COTS쏘프트웨어의 다양한 복사본들을 판매함으로써 돈을 번다. 실례로 표처리프로그람패키지에 대한 리윤은 작으며 개발비용은 가능한껏 보상할수 없다. 리윤을 얻기 위하여 1만개(혹은 지어 10만개)의 복사본들을 팔아야 한다. 이 점에서 추가적인 판매는 순리윤으로 된다. 그래서 제품이 그밖의 추가적인 류형의 하드웨어들에 쉽게 이식할수 있다면 훨씬 더 많은 돈을 벌수 있다.

물론 다른 모든 쏘프트웨어들과 마찬가지로 제품은 바로 코드는 아니다. 또한 지도서를 비롯한 문서작성문제가 있다. 표처리프로그람패키지를 다른 하드웨어에 이식하는것은 문서작성도 역시 변화시킨다는것을 의미한다. 그리하여 이식성은 령상태에서부터 새 문서를 작성하여야 할 대신에 목적으로 하는 구성배치를 반영하기 위하여 문서를 쉽게 변화시킬수 있다는것을 의미한다. 만일 완전히 새로운 제품을 개발하는것보다 눈에 익고 현존하는 제품을 새로운 콤퓨터에 이식한다면 아주 적은 숙련이 요구되게 된다. 이러한 리유로 하여 역시 이식성이 장려되여야 한다.

다음에 이식성을 실현하는 기술을 설명한다.

8．9．이식성실현기술

이식을 실현하기 위한 한가지 방도는 프로그람작성자들이 또 다른 콤퓨터에 이식할 때 문제들을 야기시킬수 있는 구성들을 리용하는것을 금지시키는것이다. 실례로 원칙은 고급언어의 표준판으로 모든 쏘프트웨어를 개발하는것이다. 그러나 이식가능한 조작체계는 어떻게 개발되는가? 결국 조작체계를 최소한의 일부 아쎔블리어코드가 없이 개발할수 있다고 하는것은 생각조차도 할수 없다. 류사하게 콤파일러는 특정한 콤퓨터에 대하여 목적코드를 생성하여야 한다. 여기서도 역시 실현의존관계의 구성요소들을 피하는것은 불가능하다.

8. 9. 1. 이식가능한 체계쏘프트웨어

거의 모든 쏘프트웨어개발을 방해할수 있는 실현의존관계측면들을 금지시키는 대신 보다 더 좋은 기법은 임의의 필요한 실현의존관계의 부분품들을 고립시키는것이다. 이 기법의 실례는 UNIX조작체계가 구성되는 방식이다[Johnson and Ritchie, 1978]. C언어로 조작체계의 약 9,000행을 작성하였다. 나머지 1,000행은 핵심부를 구성하고 있다. 핵심부는 아쎔블리어로 작성되고 매개의 실현을 위하여 다시 작성되여야 한다. C언어코드의 약 1,000행은 장치구동프로그람을 구성하고 있다. 즉 이 코드도 역시 매번 다시 작성되여야 한다. 그러나 C언어로 된 나머지 8,000행은 실현될 때마다 크게 변화되지 않았다.

체계쏘프트웨어의 이식성을 크게 하는 또 다른 유용한 기술은 추상화의 준위를 리용하는것이다(7.4.1). 실례로 도형표시부분프로그람을 생각해 보자. 사용자는 drawline 같은 지령을 원천코드에 삽입한다. 원천코드는 콤파일된 다음 도형현시부분프로그람과 련결된다. 실행시에 drawline은 사용자가 서술한대로 화면에 선을 그리도록 한다. 이것은 두개의 추상화준위를 리용하여 실현할수 있다. 고급언어로 작성된 웃준위는 사용자의 지령을 해석하고 지령을 실행시키기 위하여 적합한 낮은 준위모듈을 호출한다. 만일 도형현시부분프로그람들이 새로운 형식의 콤퓨터들에 이식된다면 사용자의 코드 혹은 도형현시부분프로그람의 웃준위에 그 어떤 변화도 없다. 그러나 부분프로그람의 아래준위모듈들은 실제적인 하드웨어와 대화하기때문에 다시 작성되여야 한다. 그리고 새로운 콤퓨터의 하드웨어는 패키지가 이전에 실현된 콤퓨터의 하드웨어와 다르다. 이 기술은 또한 ISO-OSI모형의 7개 추상화준위들에 부합되는 자료통신쏘프트웨어를 이식하는데서 성과적으로 리용되였다[Tanenbaum, 1996].

8. 9. 2. 이식가능한 응용쏘프트웨어

조작체계와 콤파일러와 같은 체계쏘프트웨어보다 응용쏘프트웨어와 관련하여 일반적으로 제품을 고급언어로 작성할수 있다. 14.1에서는 종종 실현언어와 관련하여 그 어떤 다른 선택이 없지만 어떤 언어를 선택할수 있을 때 비용 대 리득분석에 기초하여 선택을 진행하여야 한다고 서술되여 있다(5.2). 비용 대 리득분석에서 고려하여야 할 한가지 인자는 이식성에 주는 영향이다.

제품개발의 매 단계에서 보다 더 이식가능한 제품을 만들 결심들이 채택될수 있다. 실례로 일부 콤파일러들은 대문자와 소문자를 구별한다. 그러한 콤파일러들로서 This-Is-A-Name과 this-is-a-name는 다 각이한 두개의 변수로 된다. 그러나 다른 콤파일러들은 두개의 이름을 꼭 같은것으로 취급한다. 대문자와 소문자들사이의 차이에 의존하는 제품은 제품을 이식할 때 발견하기 힘든 결함들을 초래할수 있다.

흔히 프로그람작성언어에 그 어떤 선택이 없는것처럼 역시 조작체계에 그 어떤 선택이 없을수 있다. 그러나 만일 가능하다면 제품이 실행되는 조작체계는 대중적인것이여야 한다. 이것은 UNIX조작체계에 대하여 유익한 론의로 된다. UNIX는 광범한 하드웨어상에서 실현되여 왔다. 더우기 UNIX 혹은 보다 더 정확하게는 UNIX와 같은 조작체계들은 IBM/370과 VAX/VMS와 같은 대형콤퓨터조작체계상에서 실현되여 왔다. 개인용콤퓨터들

로 말하면 그것은 Linux가 가장 널리 쓰이는 조작체계인 Windows를 압도할것인가를 보아야 한다. 널리 실현된 프로그람작성언어를 리용하는것이 이식성을 도모하는것처럼 역시 널리 실현된 조작체계도 마찬가지다.

UNIX형체계에서 또 다른 체계에로 쏘프트웨어를 이식시키는것을 쉽게 하기 위하여 콤퓨터환경을 위한 이식가능한 조작체계대면부(POSIX)가 개발되였다[NIST 151, 1988]. POSIX는 응용프로그람과 UNIX조작체계사이의 대화를 규격화한다. POSIX는 응용쏘프트웨어를 거의나 문제가 없이 혹은 전혀 그 어떤 문제가 없이 이식시킬수 있는 콤퓨터들로 확장되면서 수많은 비UNIX조작체계상에서 실현되고 있다.

언어규격들은 이식성을 실현하는데서 자기의 역할을 다할수 있다. 만일 개발기업체들의 코드작성규격들이 오직 규격구성에만 리용될수 있다고 규정한다면 결과로 되는 제품은 보다 더 이식할수 있다. 이를 위하여 프로그람작성자들에게 콤파일러의 지원을 받거나 그것의 리용이 이전에 관리자측의 허가가 없이는 금지되여 온 비표준특성들의 목록을 주어야 한다. 다른 현저한 코드작성규격들처럼 이것은 기계에 의해서 검사될수 있다.

도형사용자대면부는 류사하게 표준GUI언어의 도입으로 이식가능하게 된다. 이런 실례들로서는 Motif와 X11이 있다. GUI언어들의 규격화는 10.4에서 설명한것처럼 GUI의 중요성이 커지는데 맞게 인간-콤퓨터대면부들의 이식가능성를 필요로 하는 결과를 가져 온다.

계획작성은 또한 잠재적인 앞으로의 수자적인 비호환성들을 위하여 진행되여야 한다. 실례로 만일 제품이 앞으로 32bit하드웨어상에서만 개발되고 있다면 그것은 보다 더 구식인 16bit기계에 이식되여야 할것이며 그렇게 되면 옹근수들은 ±32,767범위내에 있어야 하고 실수들의 모듈수는 $\pm 10^{68}$범위내에 있어야 한다. 더우기 정확도는 6개의 10진수자이상으로 가정되지 말아야 한다[Wallis, 1982]. 8.7.3에서 설명된것처럼 이 문제들도 Ada혹은 Java에서 제기될수 없다.

또한 필요한것은 제품이 구성되는 조작체계와 제품이 이식될수도 있는 앞으로의 조작체계들사이의 호환성이 잠재적으로 결여되도록 계획을 세우는것이다. 만일 가능하다면 조작체계호출들은 한개 혹은 두개의 모듈에 국부화되여야 한다. 임의의 사건에서 모든 조작체계호출은 조심히 문서화되여야 한다. 조작체계호출을 위한 문서작성규격은 코드를 읽게 되는 그다음의 프로그람작성자가 현재의 조작체계에 대하여 아무것도 모를것이라고 가정해야 한다. 이것은 종종 합리적인 추측으로 된다.

앞으로의 이식을 위하여 설치지도서형태로 된 문서를 제공하여야 한다. 그 지도서는 제품을 이식할 때 제품의 어느 부분을 변화시켜야 하며 어느 부분을 변화시켜야 하는가를 강조해야 할것이다. 두 실례들에서 무엇을 해야 하고 또 그것을 어떻게 할것인가 하는데 대하여 자세한 설명을 주어야 한다. 마지막으로 사용자지도서나 혹은 조작지도서와 같은 다른 지도서들에서 진행되는 변화들의 목록도 역시 설치지도서에 포함시켜야 한다.

8. 9. 3. 이식가능한 자료

하드웨어의 비호환성문제들은 8.7.1에서 지적하였다. 그러나 그러한 문제들이 해결된 다음에는 쏘프트웨어의 비호환성문제들이 남아 있게 된다. 실제로 색인순차파일의 형식

은 조작체계에 의해서 결정된다. 즉 각이한 조작체계는 일반적으로 각이한 파일형식을 가지고 있다. 많은 파일들은 그 파일안에 있는 자료의 형식과 같은 정보들이 들어 있는 머리부를 요구한다. 머리부의 형식은 거의나 항상 그 파일이 창조되는 특정한 콤파일러와 조작체계에 대하여 유일하다. 자료기지관리체계들을 리용할 때는 상황이 더 나빠질수 있다.

자료를 이식하는 가장 안전한 방법은 비구조화된(순차적인)파일을 만드는것이다. 그렇게 되면 이러한 난점은 최소로 되여 목적하는 기계에 이식될수 있다. 이러한 비구조화된 파일로부터 목적하는 구조화된 파일이 재구성될수 있다. 두개의 특수한 변화부분프로그람을 작성해야 하는데 하나는 원래의 구조화된 파일을 순차적인 형식으로 변환시키기 위하여 원천기계상에서 실행시키고 다른 하나는 이식된 순차파일로부터 구조화된 파일을 재구성하기 위하여 목적하는 기계상에서 실행시킨다. 비록 이러한 해결책이 간단한것 같지만 복잡한 자료기지모형들사이의 변환이 진행되여야 할 때 이러한 두 부분프로그람들은 결코 사소한것이 아니다.

8．10．호상조작성

이제 요구하는데 따라 인쇄하게 될 문서를 작성하려고 한다고 하자. 문서에는 매개 범주에 있는 실제적인 년간 날자별 지출과 함께 수많은 범주들로 된 년간 예산지출이 포함된다. 문서에는 또한 재정책임자로부터 정기적으로 갱신되는 통보문을 포함시켜야 한다. 이것을 실현시키는 한가지 방법으로서 예산작성을 위하여 Lotus 1-2-3과 같은 표처리프로그람을 리용하고 전체적으로 문서를 작성하기 위하여 Microsoft Word와 같은 문서처리프로그람을 리용할수도 있다. 그다음 매번 문서가 요구될 때마다 사용자는 미리 최근의 예산수자들을 반영하기 위하여 표처리프로그람을 갱신하고 그다음에는 현재의 표계산결과를 문서에 복사한다. 마지막에 사용자는 재정책임자의 통보문을 갱신하고 문서를 인쇄한다.

한가지 개선된것은 일종의 수입-수출기구를 리용하는것인데 그것은 표처리결과가 변화되는데 따라 자동적으로 문서처리프로그람이 문서에 반영되도록 하는것이다. 그러나 이것도 역시 리상적이지 못하다. 매번 사용자가 먼저 표처리프로그람을 리용하여 표계산결과를 갱신하여야 할 때마다 문서처리프로그람을 리용하여 문서를 연다. 필요한것은 문서처리프로그람과 표처리프로그람이 명백히 두개의 서로 다른 쏘프트웨어판매업체들의 호환불가능한 제품이라는데도 불구하고 그러한 두 프로그람을 통합하는 방법이다. 그러면 문서를 이전처럼 문서처리프로그람에서 연 다음 사용자는 마우스를 가지고 표처리프로그람도구를 짧게 찰칵하여 문서처리프로그람안에서 표처리프로그람을 불러 내게 된다.

이것은 호상조작성의 실례로서 각이한 판매업체들로부터 서로 다른 프로그람작성언어로 작성되고 각이한 종류의 콤퓨터환경에서 실행되고 있는 목적코드의 호상협동과정으로 정의할수 있다. 실례로 자동화된 금융기관에서 자동출납기(ATM)들의 전국적인 망을 생각해 보라. 봉사기는 하나의 기업체에 의해서 개발된 자료기지쏘프트웨어를 실행시키는 주콤퓨터이다. 의뢰자들인 ATM는 각이한 기업체들에서 개발한 C++코드를 실행시키고 있다. 더우기 통신쏘프트웨어가 있는데 여기서 기본은 안전보장이다. 이 모든 구성요소들은 ATM망이 성공적으로 기능을 수행하도록 하기 위하여 함께 동작하여야 한다.

COM과 CORBA를 비롯하여 호상조작성을 촉진시키기 위하여 수많은 표준들이 제기되였다.

8. 10. 1. COM

객체련결 및 매몰프로그람(OLE)의 첫 판본은 Windows 3.0의 한 부분으로서 1990년에 나왔다. 그것은 앞절에서 설명한 문서처리프로그람의 문서안에서 표처리프로그람과 같은 혼합문서를 지원하기 위하여 마이크로쏘프트회사가 설계하였다. 그러나 OLE 역시 호상조작성을 실현하기 위한 문제에서 부분적인 해결에 지나지 않는다는것이 인차 리해되였다. 마이크로쏘프트회사는 그다음 호상조작성이 총체적인 목적을 달성하기 위하여 보다 높은 단계인 구성요소객체모형(COM)을 개발하였다. COM은 1993년 5월에 OLE 2.0의 한 부분으로서 처음으로 나왔다. 용어 OLE는 그때 COM에 기초한 기술을 리용하여 만들어 진 구조물들을 나타내는데 리용되였다. 완전한 이름인 객체련결 및 메몰프로그람은 더는 이러한 새로운 문맥에서 뜻이 통하지 않았다. 그래서 OLE이라는 이름은 세개의 글자로 준 말로부터 본래의 이름으로 변경되였다. 1996년에 마이크로쏘프트회사는 인터네트관련기술과 관련하여 ActiveX라는 용어를 사용하기 시작하였다. 그러나 ActiveX라는 용어는 인차 COM관련기술의 두번째 의미로서 OLE와 같은 뜻을 가지는것으로 되였다. COM의 분산형판본이 분산된 콤퓨터환경하에서 호상조작성을 지원하기 위하여 1996년에 출현하였다.

호상조작성을 지원하는 기본기술은 COM기술이다. 쏘프트웨어구성요소 Q가 구성요소 P에 봉사를 제공한다고 하자. 즉 P는 Q의 의뢰자이다. P와 Q는 여러가지 각이한 방식으로 동작할수 있다. 만일 P와 Q가 같은 공정의 부분들이라고 하면 P는 Q를 호출할수 있다. 다른 한편 만일 P와 Q가 같은 기계장치상에서 실행되는 각이한 공정들에 있다면 P와 Q는 호상과정통신의 일부 형식으로 자료통신할수 있다. 그러나 만일 각이한 공정들이 망안에 있는 각이한 기계들상에서 실현되고 있다면 원격공정호출(RPC)을 리용할수 있다. COM의 리면에 있는 착상은 한개의 구성요소가 또 다른 구성요소에 봉사를 제공하게 되는 모든 정황을 위하여 공통기구를 리용하는것이다. 쏘프트웨어의 모든 부분들은 COM의 구성요소(마이크로쏘프트회사는 이것을 객체라고 하였다.)로 실현된다. 매개 구성요소는 한개 혹은 그이상의 대면부들을 가지고 있는데 그 매개의 대면부는 한개 혹은 그이상의 기능을 지원한다(이것들을 조작이라고 하였다.). COM구성요소를 리용하기 위하여 의뢰자는 구성요소의 클라스(매개 COM구성요소는 특정한 클라스의 실체이다.)와 구성요소의 특정한 대면부를 서술한 COM서고를 호출한다. 그러면 COM서고는 그 클라스의 구성요소들을 실체화하고 선택된 대면부에 지적자를 되돌려 보낸다. 이제 의뢰자가 그 대면부의 기능을 불러 내게 된다.

마이크로쏘프트회사가 명기한것처럼 COM기술은 COM이 아직 객체지향적이지 못하다는 점에서 일부 혼돈하기 쉽다. COM객체는 클라스의 실체이다. 하지만 COM은 계승을 지원하지 못한다(7.8). 따라서 COM은 객체에 기초한것이지만 객체지향적인것은 아니다. 구성요소객체모형확장 COM+는 Windows 2000과 함께 출현하였다. 이전 판본과 마찬가지로 COM+도 역시 객체에 기초한것이다. 그러나 COM의 앞으로의 판본들은 객체지향적인

것으로 될수 있다.

8. 10. 2. CORBA

1989년에 현재 객체지향기술과 관련한 약 800여명의 판매자들로 이루어 진 련합체인 객체관리그룹(OMG)이 객체지향체계들의 공통적인 구조를 개발할 목적으로 설립되였다. 특히 공통적인 객체요구중개구조(CORBA)는 분산된 환경내에서 각이한 기계장치들에서 실행되는 쏘프트웨어응용프로그람의 호상조작성을 지원한다[OMG, 1999]. 즉 CORBA는 판매자들과 망들, 언어들 그리고 조작체계들사이의 호상조작성을 지원한다. OMG표준들은 국제규격화기구의 인증을 받았으며 여기서 CORBA는 객체지향체계를 위한 국제규격으로 되였다.

CORBA에서 중심은 객체요구중개인(ORB)이며 이것은 분산형체계에서 객체가 어디에 있는가에 관계없이 의뢰자가 객체의 방법을 호출하도록 한다. 용어 미들웨어(*middleware*)는 호상조작성을 지원하는 쏘프트웨어를 설명해 주고 있다. 그리고 ORB는 《모든 의뢰자/봉사기 미들웨어의 모체》라고 불렀다[Orfali, Harkey, and Edwards, 1996]. Inprise Visibroker와 Iona ORBIX는 현재 리용할수 있는 많은 CORBA실현가운데서 두개의 CORBA로 된다.

8. 10. 3. COM과 CORBA의 비교

표면상 COM과 CORBA는 둘 다 호상조작성에 대하여 동등하게 지원하고 있다. 그러나 그 둘사이에는 많은 차이가 있다. 여기서 세가지를 고찰하기로 한다.

첫번째 차이는 COM이 세계 최대의 콤퓨터회사인 마이크로쏘프트회사의 제품이며 한편 CORBA는 말그대로 수백개의 각이한 콤퓨터기업체들에서 온 쏘프트웨어전문가들에 의해서 개발된 국제적인 표준이라는것이다. 결과 CORBA ORB제품들은 현재 COM보다 더 광범한 각이한 종류의 콤퓨터환경에서 실현되고 있다. 더우기 CORBA는 기본 통신기구들과는 별도의것으로 규정되고 있으며 반면에 COM의 통신기구는 독점적인 마이크로쏘프트회사의 제품으로 된다. 결과적으로 각이한 통신기구를 리용하는 유산적인 쏘프트웨어를 가지고 COM을 리용하는것은 어렵다.

두번째 차이는 COTS쏘프트웨어의 통합과 관련된다(2.1). 마이크로쏘프트회사는 적어도 그러한 쏘프트웨어의 80%를 제공해 주고 있다. Microsoft COTS가 마이크로쏘프트회사의 독점적인 COM으로 통합되기가 쉽다고 하는것은 현재 COTS쏘프트웨어를 CORBA에 기초한 제품으로 통합하는데 필요한 보다 복잡한 방법들에 비해서 명백한 우월성으로 된다.

세번째 차이는 CORBA는 단순하다는 우월성을 가지고 있다. COM은 가장 복잡한 마이크로쏘프트기술중의 하나이다. 즉 COM을 위한 문서는 거의 2,000페지 되며 개발자는 역시 Win32(Windows API)에 대해서 자세히 리해할것을 요구한다. 더우기 개발자들은 COM이 배우기가 어렵다는것을 알려 주었다. 다른 한편 CORBA는 200페지이하로 서술되여 있고 대부분의 CORBA제품을 위한 문서는 200～300페지정도이다.

CORBA와 COM은 각각 고유한 지지자들과 반대파들을 가지고 있다. 호상조작성이 앞으로 어떻게 실현되여 갈것인가를 보기로 한다.

8. 11. 호상조작성에 대한 앞으로의 동향

현재 두개의 주요한 호상조작성제품은 COM(8.10.1)과 CORBA(8.10.2)이다. 원래는 많은 경쟁자들이 OpenDoc에 의한 경쟁에서 현저하게 뒤떨어 졌다. 그러나 많은 다른 제품들이 현재 리용되거나 개발되고 있다. 실례로 JavaBeans는 완전히 이식가능하고 치밀하며 구조적으로 중성적이고 콤퓨터환경에 대하여 독립적인 응용프로그람작성대면부(API)로 되고 있다. Bean는 작은 규모의 구성요소이다. Enterprise bean은 독립적인 응용프로그람으로 리용할수 있거나 혹은 쏘프트웨어제품으로 통합될수 있는 보다 큰 규모의 구성요소이다. Enterprise bean들은 의뢰자-봉사기구조에서 봉사기측의 응용프로그람들을 위하여 우선적으로 리용된다[Valesky, 1999].

World Wide Web는 주요한 호상조작성의미를 가지고 있다. 한가지 선택은 의뢰자-봉사기응용프로그람과 Web에 기초한 응용프로그람들을 통합할수 있는 단일한 구조이다. 수많은 판매업체들이 이 수요에 직면하고 있다. 실례로 마이크로쏘프트회사는 분산된 인터네트구조(DNA)를 개발하였다.

CORBA가 많은 면에서 COM에 비하여 우월하다고 하는것이 제기되였다. 그럼에도 불구하고 마이크로쏘프트회사는 기업체들에 COM(혹은 집필당시에는 그의 후임인 COM+)이 호상조작성을 실현하기 위한 적합한 표준이라는것을 납득시킬수 있는 세계 최대의 콤퓨터회사로 되고 있다. CORBA나 COM이 지배적인 호상조작성기구로 될것인가 혹은 앞으로 어떤 제품이 우세를 차지하게 될것인가는 명백하지 않다. 또 다른 가능성은 두개의 경쟁적인 표준들사이의 호상조작성을 허용하게 될 CORBA와 COM의 련결이 있을수 있다는것이다.

CORBA와 COM의 지지자들사이에 경쟁이 있음에도 불구하고 두 진영은 똑같이 확장서술언어(XML)에 대하여 보다 적극적으로 나서고 있다[W3C, 2000]. XML은 초본문서술언어(HTMC)의 확장이다. HTML과 같이 XML은 일반적인 통신규약을 개발함으로써 World Wide Web상에서의 호상조작성을 촉진시키기 위하여 1994년 10월에 창설된 World Wide Web련합체(W3C)의 제품이다. W3C는 세계 각지에 400개이상의 성원기업체들을 가지고 있는 국제적인 기업체이다.

기본문제점은 XML이 메타자료언어 즉 자료들을 서술하는데 리용할수 있는 언어들이다. XML의 주요목적은 Web상에서 각이한 자료원천들의 호상조작성을 개선하는것이다. XML은 HTML이 오늘날 그러한것처럼 Web상에서 자료의 전송과 조종에서 앞으로 중요한 역할을 놀것으로 보인다.

요 약

8.1에서 재리용을 설명하였다. 8.2에서는 재리용에서 나서는 장애들에 대하여 설명한다. 재리용에 주는 객체지향파라다임의 영향에 대해서는 8.4에서 분석된다. 설계 및 실현단계에서 진행하게 되는 재리용은 8.5에서 취급하고 있다. 여기서 기본화제는 틀거리, 패

턴 그리고 쏘프트웨어구성방식이다. 유지정비에 주는 재리용의 영향에 대해서는 8.6에서 론의한다.

8.7에서는 이식성에 대하여 고찰한다. 이식성은 하드웨어(8.7.1), 조작체계(8.7.2), 수자처리쏘프트웨어(8.7.3), 콤파일러(8.7.4)에 대하여 일어 나는 비호환성의 방해를 받을수 있다. 그럼에도 불구하고 모든 제품들은 가능한껏 이식가능하게 만들기 위하여 노력하는것이 중요하다(8.8). 이식을 쉽게 진행하는 방법들로서는 일반적인 고급언어들을 리용하는것과 제품의 이식불가능한 부분들을 고립시키는 방법(8.9.1) 그리고 언어표준들에 준하는 것(8.9.2)들이 포함된다. 호상조작성에 대하여서는 8.10절에서 소개하였다. 8.10.1에서는 COM에 대하여, 8.10.2에서는 CORBA에 대하여 론의를 진행한다. 8.10.3에서는 그것들을 서로 비교한다. 8.11에서는 호상조작성에 대한 앞으로의 추세에 대하여 고찰한다.

보 충

이 장에서 재리용의 실례연구에 대한 보충적인 정보는 문헌 [Lanergan and Grasso, 1984; Matsumoto, 1984, 1987; Selby, 1989; Prieto-Díaz, 1991; Lim, 1994; Jézézquel and Meyer, 1997; and Toft, Coleman, and Ohta, 2000]에서 찾아 볼수 있다. 휼레트-패카드회사에서의 협동준위 쏘프트웨어재리용프로그람에 대해서는 [Griss, 1993]에서 설명하였다. 모토롤라에서의 두개의 비행조종사연구에 대하여서는 문헌 [Joos, 1994]에서 제시하였다. 재리용의 관리에 대하여서도 문헌 [Lim, 1998]에서 서술하였다. 재리용과 관련한 일부 경고는 문헌 [Tracz, 1995]에서 주었다. [Frakes and Fox, 1995]는 재리용에 대한 산업적인 태도에 대한 보고서이다. 객체검색과 재리용을 위한 탐색도식에 대해서는 문헌 [Isakowitz and Kauffman, 1996]에서 서술한다. Ada재리용실례연구에 대해서도 문헌 [Skazinski, 1994]에서 개괄하였다. 재리용의 비용에 대한 효과성은 문헌 [Barnes and Bollinger, 1991]에서 설명하고 문헌 [Caldiera and Basili, 1991]에서 앞으로의 재리용을 위한 구성요소들을 확증하는 방법들에 대하여 설명하였다. 문헌 [Meyer, 1996a]에서는 객체지향파라다임이 재리용을 촉진한다는 주장을 분석하였다. 재리용과 객체기술과 관련된 4개의 실례연구들은 문헌 [Fichman and Kemerer, 1997]에서 보여 주고 있다. 문헌 [Poulin, 1997]에서 재리용척도에 대하여 론의하였다. 문헌 [Prieto-Díaz, 1993]은 재리용에 관한 상태보고서이다. 재리용에 관한 연구론문들은 *IEEE Software*의 1994년 9월호에서도 찾아 볼수 있다. 쏘프트웨어공학연구소에서 진행한 구성요소에 기초한 쏘프트웨어공학기술에 관한 다양한 연구론문들은 문헌 [Brown, 1996]에서 찾아 볼수 있다. 구성요소에 기초한 쏘프트웨어공학기술에 관한 다른 론문들은 구성요소에 기초한 쏘프트웨어시험에 대해서 론의하는 문헌 [Weyuker, 1998]을 포함하여 *IEEE Software*의 1998년 9월/10월호에 있다. [Bohrer, Johnson, Nilsson, and Rubin, 1998]에서는 객체지향적업무프로그람들을 위한 구성요소들에 대하여 설명하였다. 문헌 [Mili, Addy, Mili, and Yacoub, 1999]에서는 재리용을 공학학문으로 분석하고 있다. 문헌 [Szyperski, 1998]은 구성요소지향의 쏘프트웨어에 관한 정보의 귀중한 문헌으로 되고 있다.

틀거리에 관한 귀중한 문헌은 [Lewis et al., 1995]이다. 객체지향적틀거리들의 재리용관리에 대해서는 문헌 [Sparks, Benner, and Faris, 1996]에서 설명한다. 쏘프트웨어를 구축

하기 위한 틀거리에 대해서는 문헌 [Schmid, 1996]에서 론의하였다. 개발자의 생산성에 주는 객체지향적틀거리의 영향에 대해서는 [Moser and Nierstrasz, 1996]에서 론의하였다. 문헌 [D'Souza and Wills, 1999]에서도 객체지향적틀거리들과 구성요소에 기초한 개발방법들을 제시하였다. *Communications of the ACM*의 1997년 10월호에는 문헌 [Johnson, 1997]을 비롯하여 객체지향틀거리들에 관한 수많은 기사들이 있다. 틀거리에 관한 우수한 련속기사들은 문헌 [Fayad and Johnson, 1999; Fayad and Schmidt, 1999; and Fayad, Schmidt, and Johnson, 1999]에서 찾아 볼수 있다. *Communications of the ACM*의 2000년 10월호에는 UML을 리용하여 구성요소들과 틀거리들을 모형화하는 방법을 서술한 문헌 [Kobryn, 2000]을 비롯하여 구성요소에 기초한 틀거리들에 대한 기사들이 들어 있다. 알렉싼더는 구조의 문맥안에서 설계패턴을 제기하였다[Alexander et al., 1997]. 패턴리론의 기원에 대하여 처음으로 설명한것은 문헌 [Alexander, 1999]이였다. 쏘프트웨어설계패턴들에 관한 주요한 문헌은 [Gamma, Helm, Johnson, and Vlissides, 1995]이다. 더 새로운 책은 [Vlissides, 1998]이다. [Coad, 1992]와 [Fowler, 1997a]에는 분석패턴들이 서술되여 있다. 문헌 [Schmidt, 1995]에서도 재리용가능한 객체지향통신쏘프트웨어를 개발하기 위하여 설계패턴들이 리용된데 대하여 서술한다. *Communications of the ACM*의 1996년 10월호는 설계패턴들의 우월성과 부족점을 서술한 문헌 [Cline, 1996]을 비롯하여 패턴들에 대한 많은 기사들을 실었다. 패턴들과 구조들에 관한 소론문들은 *IEEE Software* 1997년 1월/2월호에서도 보게 된다. 이러한것들로서는 패턴언어들의 중요성을 설명한 문헌 [Kerth and Cunningham, 1997]과 구조들, 패턴들 그리고 객체들을 련결한 문헌 [Monroe, Kompanek, Melton, and Garlan, 1997; Tepfenhart and Cusick, 1997; and Coplien, 1997]을 들수 있다. 설계패턴들의 리용에 관한 또 다른 론문은 [Cooper, 1998]이다. 역패턴들에 대해서는 문헌 [Brown et al., 1998]에서 설명된다.

쏘프트웨어구성방식에 관한 정보의 주요문헌은 [Shaw and Garlan, 1996]이다. 쏘프트웨어구성방식에 관한 기사들은 *IEEE Transactions on Software Engineering* 1995년 4월호와 *IEEE Software* 1995년 11월호 특히 문헌 [Shaw, 1995, and Garlan, Allen, and Ockerbloom, 1995]에서 찾아 볼수 있다. 쏘프트웨어구성방식들에 관한 더 새로운 문헌으로서는 [Bass, Clements, and Kazman, 1988; Hofmeister, Nord, and Soni, 1999; and Bosch, 2000]이 있다. 쏘프트웨어제품개발방향에 대해서도 문헌 [Lai, Weiss, and Parnas, 1999; Jazayeri; Ran, and van der Linden, 2000; and Donohoe, 2000]에서 설명하고 있다.

이식성을 실현하는 전략들에 대해서도 문헌 [Mooney, 1990]에서 찾아 볼수 있다. C와 UNIX의 이식성에 대해서는 문헌 [Johnson and Ritchie, 1978]에서 론의하였다. Ada제품들의 이식성과 관련하여서는 문헌 [Nissen and Wallis, 1984]를 참고하여야 한다.

일반적으로 호상조작성과 특히 OLE, CORBA 그리고 ActiveX에 대한 개괄을 알려면 문헌 [Adler, 1995]를 참고하면 된다. 호상조작성에 대한 정보의 종합적인 문헌은 [Orfali, Harkey, and Edwards, 1996]이다. 미들웨어에 대한 개괄은 문헌 [Brown, 1999]에 있다. COM은 문헌 [Box, 1998]에서 설명하고 있다. 자세한 정보는 마이크로쏘프트회사 홈페지 www.microsoft.com에서 얻을수 있다. CORBA에 대하여서는 문헌 [Mowbray and Zahavi, 1995]에서 자세히 설명한다. 문헌 [Leppinen, Pulkkinen, and Rautiainen, 1997]은 Java와 CORBA를 결합하기 위한 실례연구이다. *Communications of the ACM* 1998년 10월호에는

문헌 [Siegel, 1998]을 비롯하여 CORBA에 관한 기사들이 있다. Enterprise JavaBeans에 대한 자세한 취급은 문헌 [Valesky, 1999]에서 주었다.

문 제

8.1. 재리용가능성, 이식성, 호상조작성사이를 자세히 구분하시오.

8.2. 새로운 제품에서 코드모듈은 변화되지 않은 상태에서 재리용된다. 이 재리용이 어떤 방법들로 제품의 총체적인 비용을 줄이는가? 비용은 어떤 방법들에서 변화되지 않는가?

8.3. 코드모듈이 한가지 변화 즉 더하기연산에서 덜기연산으로 변화되면서 재리용된다고 가정하자. 이 부차적인 변화가 문제 8.2의 비용절약에 영향을 미치는가?

8.4. 재리용가능성에 주는 응집도의 영향은 무엇인가?

8.5. 재리용가능성에 주는 결합도의 영향은 무엇인가?

8.6. 당신은 이제 다양한 오염조종제품들을 개발하는 큰 기업체에 가입하였다. 그 기업체는 9,500개의 각이한 FORTRAN모듈들로 이루어 진 수백개의 쏘프트웨어제품들을 가지고 있다. 당신은 앞으로의 제품들에서 이 모듈들을 가능한껏 재리용하기 위한 계획을 제의하였다. 당신의 제안은 무엇인가?

8.7. 자동서고순환체계를 생각해 보자. 모든 책들에는 막대부호가 있으며 빌리는 사람들도 모두 막대부호코드가 있는 카드를 가지고 있다. 빌리러 오는 사람이 책을 빌릴 때 사서는 책에 있는 막대부호들과 빌리러 온 사람의 카드를 자세히 조사하고 콤퓨터말단장치에 C를 입력한다. 그와 류사하게 책을 바칠 때 다시 그것을 자세히 보고 사서는 R를 입력한다. 사서들은 책들을 서고에 보충해 넣거나 서고에서 없애 버린다. 빌리는 사람들은 말단장치에 가서 특정한 저자의 이름을 가지고 도서관의 모든 책들을 확정할수 있으며(빌리러 온 사람이 A를 입력하면 저자의 이름이 뒤따라 나온다.) 모든 책들은 특정한 제목(T를 입력하면 제목이 나온다.)을 가지고 확정하거나 혹은 특정한 주제로부터 모든 책들을 확정할수 있다(S를 입력하면 주제가 뒤따라 나온다.). 끝으로 만일 빌리려는 사람이 현재 어떤 책을 빌리고 싶으면 사서는 책을 바칠 때 그것을 요구한 사람을 위하여 보관하도록 책에 표식을 할수 있다(H를 입력하면 책의 번호가 나온다.). 당신이 어떻게 재리용가능한 모듈들에 대하여 높은 비률을 보장하겠는가를 설명하시오.

8.8. 당신은 은행보고서가 정확한가를 결정하기 위하여 제품을 개발해 줄것을 부탁받았다. 필요한 자료는 월초의 잔고, 번호, 날자, 매개 행표의 량, 매개 예금날자와 량, 월말의 잔고들이 들어 있다. 당신이 제품의 가능한껏 많은 모듈들이 앞으로의 제품들에서 재리용될수 있다는것을 어떻게 담보하겠는가 하는것을 설명하시오.

8.9. 금융기관의 자동출납기(ATM)를 생각해 보자. 사용자는 카드를 넣고 4자리수자로 된 개인식별부호(PIN)을 입력한다. 만일 개인식별부호가 정확하면 카드가 나온다. 그렇지 않으면 사용자는 4개의 서로 다른 은행계산서에 대한 다음의 조작들을 진행할수 있다.

(1) 일정한 량을 예금하라. 날자, 예금량 그리고 계산서번호를 보여 주는 령수증을 인쇄한다.

(2) 20딸라의 단위로 200딸라까지 되찾으시오(계산서는 찾아 가지 못할수도 있다.). 돈과 함께 사용자에게 날자, 되찾아 간 량, 계산서번호 그리고 돈을 찾아 간 다음 계산잔고를 보여 주는 령수증을 준다.

(3) 계산잔고를 확정하시오. 이것은 화면에 표시된다.

(4) 두 계산서들사이에 자금을 옮겨 주시오. 다시 자금이 옮겨 간 계산서에는 돈을 지내 많이 찾아 가지 말도록 하여야 한다. 사용자에게 날자와 넘겨 준 량 그리고 두 계산서번호들을 보여 주는 령수증을 준다.

(5) 끝내기. 카드가 나온다.

제품이 가능한껏 많은 모듈을 앞으로의 제품들에서 재리용할수 있다는것을 어떻게 담보하겠는가를 설명하시오.

8.10. 개발자들이 아란 5호 쏘프트웨어에 있는 고장을 쏘프트웨어생명주기에서 어떻게 일찌기 찾을수 있겠는가(8.3.6)?

8.11. 8.5.2에는 《1970년대의 레이손 COBOL프로그람론리구조가 오늘날의 객체지향응용프로그람틀거리의 고전적인 조상이다.》라고 언급되여 있다. 기술이전에서 이것은 무엇을 의미하는가.

8.12. 그림 8-3의 설계패턴에서 추상클라스들이 노는 역할을 설명하시오.

8.13. 자동서고순환체계(문제 8.7)가 가능한껏 이식가능하다는것을 어떻게 담보하겠는가를 설명하시오.

8.14. 행보고서가 정확한가(문제 8.8)하는것을 검사하는 제품이 가능한껏 이식가능하다는것을 어떻게 담보하겠는가를 설명하시오.

8.15. 문제 8.9의 금융기관의 자동출납기(ATM)를 위한 쏘프트웨어가 가능한껏 이식가능하다는것을 어떻게 담보하겠는가를 설명하시오.

8.16. 당신의 기업체는 암치료에 쓰일 새로운 형태의 레이자용 실시간조종체계를 개발하고 있다. 당신은 두개의 아쎔블리어모듈들을 작성하는것을 책임지고 있다. 당신은 결과로 되는 코드가 가능한껏 이식가능하다는것을 담보하도록 어떻게 팀을 지도하겠는가?

8.17. 당신은 75만행되는 COBOL제품을 회사의 새 콤퓨터에 이식할 책임을 지고 있다. 당신이 원천코드를 새 기계에 복사하고 그것을 콤파일하려고 노력하였지만 1만 5천개이상의 입출력명령문들가운데서 그 매개가 새로운 콤파일러가 처리할수 없는 비표준 COBOL문법으로 씌여 져 있다는것을 발견하였다. 이제 당신은 무엇을 하는가?

8.18. (과정안상 목표) 부록 1의 브로드랜즈지역아동병원제품은 구조화파라다임을 리용하여 개발되였다고 가정하자. 제품의 어느 부분이 앞으로의 제품들에서 재리용될수 있겠는가? 이제는 제품이 객체지향파라다임을 리용하여 개발되였다고 가정하자. 제품의 어느 부분이 앞으로의 제품들에서 재리용될수 있겠는가?

8.19. (쏘프트웨어공학독본) 교원은 문헌 [Schricker, 2000]의 복사본들을 나누어 줄것이다. 지난 40년동안 COBOL은 가장 널리 쓰이는 프로그람작성언어로 되여 왔다. 그 객체지향적 COBOL이 다같이 어디서나 볼수 있을것이라고 생각하는가?

참 고 문 헌

[AdaIC, 1997] "DoD to Replace Ada Mandate with Software-Engineering Process," *AdaIC News* (Summer 1997), Ada Information Clearinghouse, Falls Church, VA.

[Adler, 1995] R. M. ADLER, "Emerging Standards for Component Software," *IEEE Computer* **28** (March 1995), pp. 68–77.

[Alexander, 1999] C. ALEXANDER, "The Origins of Pattern Theory," *IEEE Software* **16** (September/October 1999), pp. 71–82.

[Alexander et al., 1977] C. ALEXANDER, S. ISHIKAWA, M. SILVERSTEIN, M. JACOBSON, I. FIKSDAHL-KING, AND S. ANGEL, *A Pattern Language*, Oxford University Press, New York, 1977.

[Anderson et al., 1995] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, AND D. SORENSEN, *LAPACK Users' Guide*, 2nd ed, SIAM, Philadelphia, 1995.

[ANSI X3.159, 1989] "The Programming Language C," ANSI X3.159-1989, American National Standards Institute, New York, 1989.

[ANSI/IEEE 754, 1985] "Standard for Binary Floating Point Arithmetic," ANSI/IEEE 754, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1985.

[ANSI/IEEE 770X3.97, 1983] "Pascal Computer Programming Language," ANSI/IEEE 770X3.97-1983, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1983.

[ANSI/MIL-STD-1815A, 1983] "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A, American National Standards Institute, United States Department of Defense, Washington, DC, 1983.

[Barnes and Bollinger, 1991] B. H. BARNES AND T. B. BOLLINGER, "Making Reuse Cost-Effective," *IEEE Software* **8** (January 1991), pp. 13–24.

[Bass, Clements, and Kazman, 1998] L. BASS, P. CLEMENTS, AND R. KAZMAN, *Software Architecture in Practice*, Addison-Wesley, Reading, MA, 1998.

[Bohrer, Johnson, Nilsson, and Rubin, 1998] K. BOHRER, V. JOHNSON, A. NILSSON, AND B. RUBIN, "Business Process Components for Distributed Object Applications," *Communications of the ACM* **41** (June 1998), pp. 43–48.

[Bosch, 2000] J. BOSCH, *Design and Use of Software Architectures*, Addison-Wesley, Reading, MA, 2000.

[Box, 1998] D. BOX, *Essential COM*, Addison-Wesley, Reading, MA, 1998.

[Brown, 1996] A. W. BROWN, *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*, IEEE Computer Society Press, Los Alamitos, CA, 1996.

[Brown, 1999] A. W. BROWN, "Mastering the Middleware Muddle," *IEEE Software* **16** (July/August 1999), pp. 18–21.

[Brown et al., 1998] W. J. BROWN, R. C. MALVEAU, W. H. BROWN, H. W. MCCORMICK, III, AND T. J. MOWBRAY, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley and Sons, New York, 1998.

[Caldiera and Basili, 1991] G. CALDIERA AND V. R. BASILI, "Identifying and Qualifying Reusable Software Components," *IEEE Computer* **24** (February 1991), pp. 61–70.

[Carlson, Druffel, Fisher, and Whitaker, 1980] W. E. CARLSON, L. E. DRUFFEL, D. A. FISHER, AND W. A. WHITAKER, "Introducing Ada," *Proceedings of the ACM Annual Conference, ACM 80*, Nashville, TN, 1980, pp. 263–71.

[Cline, 1996] M. P. CLINE, "The Pros and Cons of Adopting and Applying Design Patterns in the Real World," *Communications of the ACM* **39** (October 1996), pp. 47–49.

[Coad, 1992] P. COAD, "Object-Oriented Patterns," *Communications of the ACM* **35** (September 1992), pp. 152–59.

[Cooper, 1998] J. W. COOPER, "Using Design Patterns," *Communications of the ACM* **42** (June 1998), pp. 65–68.

[Coplien, 1997] J. O. COPLIEN, "Idioms and Patterns as Architectural Literature," *IEEE Software* **14** (January/February 1997), pp. 36–42.

[Dongarra, Pozo, and Walker, 1993] J. DONGARRA, R. POZO, AND D. WALKER, "LAPACK++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra," *Proceedings of Supercomputing '93,* IEEE Press, New York, 1993, pp. 162–71.

[Donohoe, 2000] P. DONOHOE (EDITOR), *Software Product Lines: Experience and Research Directions,* Kluwer Academic Publishers, Boston, 2000.

[D'Souza and Wills, 1999] D. D'SOUZA AND A. WILLS, *Objects, Components, and Frameworks with UML: The Catalysis Approach,* Addison-Wesley, Reading, MA, 1999.

[Fayad and Johnson, 1999] M. FAYAD AND R. JOHNSON, *Domain-Specific Application Frameworks: Frameworks Experience by Industry,* John Wiley and Sons, New York, 1999.

[Fayad and Schmidt, 1999] M. FAYAD AND D. C. SCHMIDT, *Building Application Frameworks: Object-Oriented Foundations of Framework Design,* John Wiley and Sons, New York, 1999.

[Fayad, Schmidt, and Johnson, 1999] M. FAYAD, D. C. SCHMIDT AND R. JOHNSON, *Implementing Application Frameworks: Object-Oriented Frameworks at Work,* John Wiley and Sons, New York, 1999.

[Fichman and Kemerer, 1997] R. G. FICHMAN AND C. F. KEMERER, "Object Technology and Reuse: Lessons from Early Adopters," *IEEE Computer* **30** (July 1997), pp. 47–57.

[Fisher, 1976] D. A. FISHER, "A Common Programming Language for the Department of Defense—Background and Technical Requirements," Report P-1191, Institute for Defense Analyses, Alexandria, VA, 1976.

[Flanagan and Loukides, 1997] D. FLANAGAN AND M. LOUKIDES, *Java in a Nutshell: A Desktop Quick Reference,* 2nd ed., O'Reilly and Associates, Sebastopol, CA, 1997.

[Fowler, 1997a] M. FOWLER, *Analysis Patterns: Reusable Object Models,* Addison-Wesley, Reading, MA, 1997.

[Frakes and Fox, 1995] W. B. FRAKES AND C. J. FOX, "Sixteen Questions about Software Reuse," *Communications of the ACM* **38** (June 1995), pp. 75–87.

[Gamma, Helm, Johnson, and Vlissides, 1995] E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, MA, 1995.

[Garlan, Allen, and Ockerbloom, 1995] D. GARLAN, R. ALLEN, AND J. OCKERBLOOM, "Architectural Mismatch: Why Reuse Is So Hard," *IEEE Software* **12** (November 1995), pp. 17–26.

[Gifford and Spector, 1987] D. GIFFORD AND A. SPECTOR, "Case Study: IBM's System/360-370 Architecture," *Communications of the ACM* **30** (April 1987), pp. 292–307.

[Green, 2000] P. GREEN, "FW: Here's an Update to the Simulated Kangaroo Story," *The Risks Digest* **20** (January 23, 2000), catless.ncl.ac.uk/Risks/20.76.html.

[Griss, 1993] M. L. GRISS, "Software Reuse: From Library to Factory," *IBM Systems Journal* **32** (No. 4, 1993), pp. 548–66.

[Hofmeister, Nord, and Soni, 1999] C. HOFMEISTER, R. NORD, AND D. SONI, *Applied*

Software Architecture, Addison-Wesley, Reading, MA, 1999.
[Holzner, 1993] S. HOLZNER, *Microsoft Foundation Class Library Programming,* Brady, New York, 1993.
[Isakowitz and Kauffman, 1996] T. ISAKOWITZ AND R. J. KAUFFMAN, "Supporting Search for Reusable Software Objects," *IEEE Transactions on Software Engineering* **22** (June 1996), pp. 407–23.
[ISO-7185, 1980] "Specification for the Computer Programming Language Pascal," ISO-7185, International Standards Organization, Geneva, 1980.
[ISO/IEC 1539–1, 1997] "Information Technology—Programming Languages—Fortran—Part 1: Base Language," ISO/IEC 1539–1, International Standards Organization, International Electrotechnical Commission, Geneva, 1997.
[ISO/IEC 1989, 2000] "Information Technology—Programming Languages, Their Environments and System Software Interfaces—Programming Language COBOL," Committee Draft 1.8, Proposed Revision of ISO 1989:1985, NCITS/J4 and ISO/IEC JCT1/SC22/WG4, International Organization for Standardization, International Electrotechnical Commission, Geneva, February 2000.
[ISO/IEC 8652, 1995] "Programming Language Ada: Language and Standard Libraries," ISO/IEC 8652, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.
[ISO/IEC 14882, 1998] "Programming Language C++," ISO/IEC 14882, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1998.
[Jazayeri, Ran, and van der Linden, 2000] M. JAZAYERI, A. RAN, AND F. VAN DER LINDEN, *Software Architecture for Product Families: Principles and Practice,* Addison-Wesley, Reading, MA, 2000.
[Jensen and Wirth, 1975] K. JENSEN AND N. WIRTH, *Pascal User Manual and Report,* 2nd ed., Springer-Verlag, New York, 1975.
[Jézéquel and Meyer, 1997] J.-M. JÉZÉQUEL AND B. MEYER, "Put It in the Contract: The Lessons of Ariane," *IEEE Computer* **30** (January 1997), pp. 129–30.
[Johnson, 1979] S. C. JOHNSON, "A Tour through the Portable C Compiler," 7th ed., *UNIX Programmer's Manual,* Bell Laboratories, Murray Hill, NJ, January 1979.
[Johnson, 1997] R. E. JOHNSON, "Frameworks = (Components + Patterns)," *Communications of the ACM* **40** (October 1997), pp. 39–42.
[Johnson and Ritchie, 1978] S. C. JOHNSON AND D. M. RITCHIE, "Portability of C Programs and the UNIX System," *Bell System Technical Journal* **57** (No. 6, Part 2, 1978), pp. 2021–48.
[Jones, 1984] T. C. JONES, "Reusability in Programming: A Survey of the State of the Art," *IEEE Transactions on Software Engineering* **SE-10** (September 1984), pp. 488–94.
[Jones, 1996] C. JONES, *Applied Software Measurement,* McGraw-Hill, New York, 1996.
[Joos, 1994] R. JOOS, "Software Reuse at Motorola," *IEEE Software* **11** (September 1994), pp. 42–47.
[Kernighan and Ritchie, 1978] B. W. KERNIGHAN AND D. M. RITCHIE, *The C Programming Language,* Prentice Hall, Englewood Cliffs, NJ, 1978.
[Kerth and Cunningham, 1997] N. L. KERTH AND W. CUNNINGHAM, "Using Patterns to Improve Our Architectural Vision," *IEEE Software* **14** (January/February 1997), pp. 53–59.
[Klatte et al., 1993] R. KLATTE, U. KULISCH, A. WIETHOFF, C. LAWO, AND M. RAUCH,

C-XSC: A C++ Class Library for Extended Scientific Computing, Springer-Verlag, Heidelberg, Germany, 1993.

[Kobryn, 2000] C. KOBRYN, "Modeling Components and Frameworks with UML," *Communications of the ACM* **43** (October 2000), pp. 31–38.

[Lai, Weiss, and Parnas, 1999] C. T. R. LAI, D. M. WEISS, AND D. L. PARNAS, *Software Product-Line Engineering: A Family-Based Software Development Process,* Addison-Wesley, Reading, MA, 1999.

[Lanergan and Grasso, 1984] R. G. LANERGAN AND C. A. GRASSO, "Software Engineering with Reusable Designs and Code," *IEEE Transactions on Software Engineering* **SE-10** (September 1984), pp. 498–501.

[Langtangen, 1994] H. P. LANGTANGEN, "Getting Started with Finite Element Programming in DIFFPACK," The DiffPack Report Series, SINTEF, Oslo, Norway, October 2, 1994.

[Leppinen, Pulkkinen, and Rautiainen, 1997] M. LEPPINEN, P. PULKKINEN, AND A. RAUTIAINEN, "Java- and CORBA-Based Network Management," *IEEE Computer* **30** (June 1997), pp. 83–87.

[Lewis et al., 1995] T. LEWIS, L. ROSENSTEIN, W. PREE, A. WEINAND, E. GAMMA, P. CALDER, G. ANDERT, J. VLISSIDES, AND K. SCHMUCKER, *Object-Oriented Application Frameworks,* Manning, Greenwich, CT, 1995.

[Lim, 1994] W. C. LIM, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software* **11** (September 1994), pp. 23–30.

[Lim, 1998] W. C. LIM, *Managing Software Reuse,* Prentice Hall, Upper Saddle River, NJ, 1998.

[Liskov, Snyder, Atkinson, and Schaffert, 1977] B. LISKOV, A. SNYDER, R. ATKINSON, AND C. SCHAFFERT, "Abstraction Mechanisms in CLU," *Communications of the ACM* **20** (August 1977), pp. 564–76.

[Mackenzie, 1980] C. E. MACKENZIE, *Coded Character Sets: History and Development,* Addison-Wesley, Reading, MA, 1980.

[Matsumoto, 1984] Y. MATSUMOTO, "Management of Industrial Software Production," *IEEE Computer* **17** (February 1984), pp. 59–72.

[Matsumoto, 1987] Y. MATSUMOTO, "A Software Factory: An Overall Approach to Software Production," in: *Tutorial: Software Reusability,* P. Freeman (Editor), Computer Society Press, Washington, DC, 1987, pp. 155–78.

[Meyer, 1987] B. MEYER, "Reusability: The Case for Object-Oriented Design," *IEEE Software* **4** (March 1987), pp. 50–64.

[Meyer, 1990] B. MEYER, "Lessons from the Design of the Eiffel Libraries," *Communications of the ACM* **33** (September 1990), pp. 68–88.

[Meyer, 1996a] B. MEYER, "The Reusability Challenge," *IEEE Computer* **29** (February 1996), pp. 76–78.

[Mili, Addy, Mili, and Yacoub, 1999] A. MILI, E. ADDY, H. MILI, AND S. YACOUB, "Toward an Engineering Discipline of Software Reuse," *IEEE Software* **16** (September/October 1999), pp. 22–31.

[Monroe, Kompanek, Melton, and Garlan, 1997] R. T. MONROE, A. KOMPANEK, R. MELTON, AND D. GARLAN, "Architectural Styles, Design Patterns, and Objects," *IEEE Software* **14** (January/February 1997), pp. 43–52.

[Mooney, 1990] J. D. MOONEY, "Strategies for Supporting Application Portability," *IEEE Computer* **23** (November 1990), pp. 59–70.

[Moser and Nierstrasz, 1996] S. MOSER AND O. NIERSTRASZ, "The Effect of Object-Oriented Frameworks on Developer Productivity," *IEEE Computer* **29** (September 1996), pp. 45–51.

[Mowbray and Zahavi, 1995] T. J. MOWBRAY AND R. ZAHAVI, *The Essential CORBA,* John Wiley and Sons, New York, 1995.
[Musser and Saini, 1996] D. R. MUSSER AND A. SAINI, *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library,* Addison-Wesley, Reading, MA, 1996.
[Nissen and Wallis, 1984] J. NISSEN AND P. WALLIS (EDITORS), *Portability and Style in Ada,* Cambridge University Press, Cambridge, U.K., 1984.
[NIST 151, 1988] "POSIX: Portable Operating System Interface for Computer Environments," Federal Information Processing Standard 151, National Institute of Standards and Technology, Washington, DC, 1988.
[Norusis, 2000] M. J. NORUSIS, *SPSS 10.0 Guide to Data Analysis,* Prentice Hall, Upper Saddle Valley River, NJ, 2000.
[OMG, 1999] "The Common Object Request Broker: Architecture and Specification. Revision 2.3.1," Document 99.10.07, Object Management Group, Needham, MA, October 1999.
[Orfali, Harkey, and Edwards, 1996] R. ORFALI, D. HARKEY, AND J. EDWARDS, *The Essential Distributed Objects Survival Guide,* John Wiley and Sons, New York, 1996.
[Phillips, 1986] J. PHILLIPS, *The NAG Library: A Beginner's Guide,* Clarendon Press, Oxford, U.K., 1986.
[Poulin, 1997] J. S. POULIN, *Measuring Software Reuse: Principles, Practice, and Economic Models,* Addison-Wesley, Reading, MA, 1997.
[Prieto-Díaz, 1991] R. PRIETO-DÍAZ, "Implementing Faceted Classification for Software Reuse," *Communications of the ACM* **34** (May 1991), pp. 88–97.
[Prieto-Díaz, 1993] R. PRIETO-DÍAZ, "Status Report: Software Reusability," *IEEE Software* **10** (May 1993), pp. 61–66.
[Schach, 1992] S. R. SCHACH, *Software Reuse: Past, Present, and Future,* videotape, 150 mins, US-VHS format. IEEE Computer Society Press, Los Alamitos, CA, November 1992.
[Schach, 1994] S. R. SCHACH, "The Economic Impact of Software Reuse on Maintenance," *Journal of Software Maintenance—Research and Practice* **6** (July/August 1994), pp. 185–96.
[Schach, 1997] S. R. SCHACH, *Software Engineering with Java,* Richard D. Irwin, Chicago, 1997.
[Schmid, 1996] H. A. SCHMID, "Creating Applications from Components: A Manufacturing Framework Design," *IEEE Software* **13** (November/December 1996), pp. 67–75.
[Schmidt, 1995] D. C. SCHMIDT, "Using Design Patterns to Develop Reusable Object-Oriented Communications Software," *Communications of the ACM* **38** (October 1995), pp. 65–74.
[Schricker, 2000] D. SCHRICKER, "Cobol for the Next Millennium," *IEEE Software* **17** (March/April 2000), pp. 48–52.
[Selby, 1989] R. W. SELBY, "Quantitative Studies of Software Reuse," in: *Software Reusability. Volume 2. Applications and Experience,* T. J. Biggerstaff and A. J. Perlis (Editors), ACM Press, New York, 1989, pp. 213–33.
[Shaw, 1995] M. SHAW, "Comparing Architectural Design Styles," *IEEE Software* **12** (November 1995), pp. 27–41.
[Shaw and Garlan, 1996] M. SHAW AND D. GARLAN, *Software Architecture: Perspectives on an Emerging Discipline,* Prentice Hall, Upper Saddle Valley River, NJ, 1996.
[Siegel, 1998] J. SIEGEL, "OMG Overview: CORBA and the OMA in Enterprise Computing," *Communications of the ACM* **41** (October 1998), pp. 37–43.

[Skazinski, 1994] J. G. SKAZINSKI, "Porting Ada: A Report from the Field," *IEEE Computer* **27** (October 1994), pp. 58–64.

[Sparks, Benner, and Faris, 1996] S. SPARKS, K. BENNER, AND C. FARIS, "Managing Object-Oriented Framework Reuse," *IEEE Computer* **29** (September 1996), pp. 52–61.

[Szyperski, 1998] C. SZYPERSKI, *Component Software: Beyond Object-Oriented Programming,* Addison-Wesley, Reading, MA, 1998.

[Tanenbaum, 1996] A. S. TANENBAUM, *Computer Networks,* 3rd ed., Prentice Hall, Upper Saddle River, NJ, 1996.

[Tepfenhart and Cusick, 1997] W. M. TEPFENHART AND J. J. CUSICK, "A Unified Object Topology," *IEEE Software* **14** (January/February 1997), pp. 31-35.

[Toft, Coleman, and Ohta, 2000] P. TOFT, D. COLEMAN, AND J. OHTA, "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," in: *Software Product Lines: Experience and Research Directions,* P. Donohoe (Editor), Kluwer Academic Publishers, Boston, 2000, pp. 111–132.

[Tracz, 1994] W. TRACZ, "Software Reuse Myths Revisited," *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 271–72.

[Tracz, 1995] W. TRACZ, *Confessions of a Used Program Salesman: Institutionalizing Software Reuse,* Addison-Wesley, Reading, MA, 1995.

[Valesky, 1999] T. C. VALESKY, *Enterprise JavaBeans: Developing Component-Based Distributed Applications,* Addison-Wesley, Reading, MA, 1999.

[Vlissides, 1998] J. VLISSIDES, *Pattern Hatching: Design Patterns Applied,* Addison-Wesley, Reading, MA, 1998.

[W3C, 2000] "Canonical XML, Version 1.0," W3C Working Draft, World Wide Web Consortium, Massachusetts Institute of Technology Laboratory for Computer Science, Boston MA; Institut National de Recherche en Informatique et en Automatique, France; Keio University, Shonan Fujisawa Campus, Japan, June 1, 2000.

[Wallis, 1982] P. J. L. WALLIS, *Portable Programming,* John Wiley and Sons, New York, 1982.

[Wegner, 1992] P. WEGNER, "Dimensions of Object-Oriented Modeling," *IEEE Computer* **25** (October 1992), pp. 12–20.

[Wells, 1996] T. D. WELLS, "A Technical Comparison of Borland ObjectWindows 2.0 and Microsoft MFC 2.5," www.it.rit.edu/~tdw/refs/om.htm, February 5, 1996.

[Weyuker, 1998] E. J. WEYUKER, "Testing Component-Based Software: A Cautionary Tale," *IEEE Software* **15** (September/October 1998), pp. 54–59.

[Wilson, Rosenstein, and Shafer, 1990] D. A. WILSON, L. S. ROSENSTEIN, AND D. SHAFER, *Programming with MacApp,* Addison-Wesley, Reading, MA, 1990.

제9장. 계획작성과 타산*)

쏘프트웨어제품을 개발하는데서 나서는 난점들을 해결하는것은 쉬운 일이 아니다. 큰 쏘프트웨어제품을 완성하는데는 시간과 자원이 요구된다. 다른 큰 프로젝트구성과 마찬가지로 프로젝트초기단계에서 주의깊게 계획을 작성하는것은 성공과 실패를 좌우하는 가장 중요한 요인으로 된다. 이러한 초기의 계획작성은 결코 충분한것이 아니다. 시험과 마찬가지로 계획작성은 쏘프트웨어제품개발과 유지정비과정전반에 걸쳐 계속 진행하여야 한다. 계획작성을 계속 진행하여야 하는데도 불구하고 이러한 활동들은 명세서를 작성한 다음 즉시에 시작되며 설계에 들어 가기전에 계획작성이 진행되여야 한다. 공정의 이 시점에서 개발기간과 비용이 타산되고 프로젝트를 완성하기 위한 구체적인 계획이 작성된다.

이장에서는 이러한 두가지 형태의 계획작성 즉 프로젝트전반에 걸쳐 진행되는 계획작성과 일단 명세서가 완성된 다음에 수행되여야 하는 적극적인 계획작성으로 나누어 고찰하도록 한다.

9. 1. 계획작성과 쏘프트웨어개발공정

리상적으로 개발공정의 초시기에 전체적인 쏘프트웨어프로젝트를 계획하고 목적하는 쏘프트웨어가 최종적으로 의뢰자에게 배포될 때까지 그 계획에 따라야 한다. 그러나 이것은 초기 단계에서는 완전한 제품을 만들기 위한 계획을 작성하는데 필요한 정보가 충분하지 못한데로부터 불가능하다. 실례로 요구사항확정단계에서 그 어떤 계획작성(요구사항확정단계 그자체를 위하여보다도 기타 다른)을 하는것은 헛된 일이다.

개발자들은 신속원형을 작성하였고(3.3) 의뢰자들은 신속원형이 목적하는 제품의 주요한 기능들을 포괄하고 있다는것을 인정했다고 하자. 이 단계에서는 그 제품을 위한 정확한 기간과 비용타산을 합리적으로 할수 있을것 같다. 공교롭게도 그것은 맞지 않는다. 요구사항확정단계의 마감과 명세작성단계의 마감에서 개발자들의 처분에 대한 정보들사이의 차이는 륜곽적으로 대충설계하는것과 자세하게 설계하는것사이의 차이와 류사한것으로 볼수 있다. 요구사항확정단계의 마감까지는 개발자들은 기껏해서 의뢰자들이 요구하는것을 비형식적으로 리해하게 된다. 이와 대비적으로 명세작성단계의 마감에 즉 의뢰자들이 구축하려고 하는것을 정확히 서술한 문서에 수표를 하는 때에 개발자들은 목적하는 제품의 모든 측면들을 자세하게 리해하게 된다. 이것은 정확한 개발기간과 비용타산이 결정될수 있는 공정에서의 시작으로 된다.

그럼에도 불구하고 일부 경우에 기업체들은 명세서가 작성되기전에 개발기간과 비용

---

*) 앞에서 설명한바와 같이 이 장에서 자료는 2개의 부분을 나란히 고찰한다. 9장에서의 자료는 실례연구(11.15과 12.8, 문제 11.20과 문제 11.21)의 쏘프트웨어프로젝트관리계획과 파정안상 도달목표(문제 11.16)를 위하여 요구된다.

타산을 진행할것을 요구한다. 극단한 경우에 의뢰자는 지어 신속원형이 작성되기전에 한두시간정도 예비적으로 론의를 진행한 기초우에서 값을 흥정한다. 그림 9-1은 이것이 무엇을 의미하는가를 보여 주고 있다.

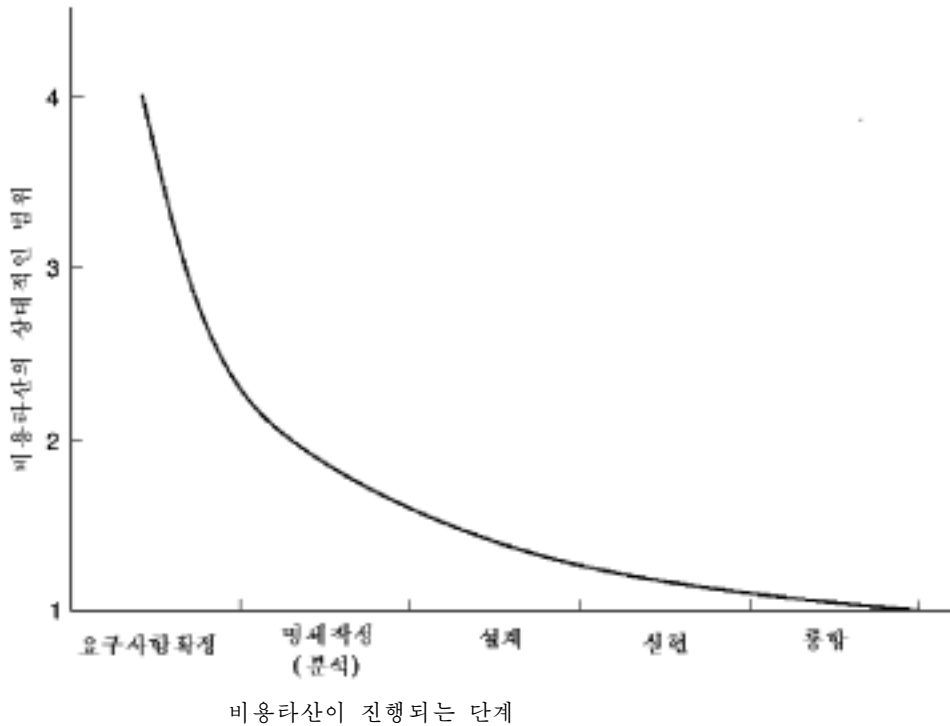



그림 9-1. 매 생명주기단계의 비용타산에 대한 상대적인 범위의 모형평가

문헌 [Boehm et al., 2000]에 있는 모형에 기초하여 생명주기의 여러 단계들에서 요구되는 비용의 상대적인 범위를 보여 주고 있다. 실례로 어떤 제품이 통합단계의 마감에서 시험에 통과되여 의뢰자에게 배포되였을 때 그 비용이 백만딸라였다고 가정하자. 비용타산이 요구사항확정단계에서는 도중에 있다고 하면 그것은 25만내지 4백만딸라정도의 범위에 있게 될것이다. 류사하게 명세작성단계에서 비용타산이 도중에 진행된다고 하면 50만내지 2백만딸라정도의 범위로 줄어 들게 될것이다. 더우기 비용타산이 명세작성단계의 마감에서 즉 가장 가능한 정확한 시기에 진행되였다고 하면 결과는 67만내지 150만딸라정도의 범위에 있게 될것이다. 달리말하면 비용타산은 다음 절에서 그 리유를 밝히지만 정확히 과학적이지 못하다는것이다. 이 모형이 기초한 자료는 미공군전자체계에서 제기한 5개의 제안[Devenny, 1976]을 비롯하여 낡은것들이고 타산방법은 그시기부터 개선되기 시작하였다. 그럼에도 불구하고 그림 9-1에 있는 곡선의 전반적인 형태는 아마 심하게 변화되지는 않는다. 결과적으로 기간과 비용을 때이르게 타산하는것 다시말하여 명세서가 의뢰자들에 의해서 승인을 받기전에 타산을 진행하는것은 적합한 시기에 타산을 진행하는것보다 정확도가 상당히 떨어 진다.

이제 기간과 비용을 타산하는 방법들에 대하여 검토하기로 한다. 이 장의 나머지부

분들에서는 명세작성단계가 완료되고 실제적인 타산과 계획작성을 진행할수 있게 되여 있다고 가정한다.

9. 2. 기간과 비용의 타산

예산은 쏘프트웨어프로젝트관리계획에서 필수불가결한 부분이다. 개발을 시작하기전에 의뢰자는 제품을 개발하는데 얼마만한 비용이 드는가를 알려고 한다. 개발팀이 실제의 비용을 과소평가하면 개발자들은 프로젝트에 대하여 돈을 잃게 될수 있다. 다른 한편 과대평가하면 의뢰자들은 비용 대 리득분석이나 투자에 대한 보수에 기초하여 제품개발에서 리익이 없다고 생각할수도 있다. 두 경우로 보아 정확하게 비용을 타산하는것이 중요하다는것이 명백하다.

사실상 비용에는 두가지 형태가 있는데 그것은 쏘프트웨어개발과 관련된다. 하나는 개발자들에 대한 비용으로서 내적인 비용(*internal cost*)이 있고 다른 하나는 의뢰자에 대한 비용으로서 외적인 비용(*external cost*)이 있다. 내적인 비용은 프로젝트에 참가하는 개발팀과 관리자, 보장성원들의 로임과 그리고 제품개발에 필요한 하드웨어와 쏘프트웨어의 비용, 세금이나 리익, 관리자들의 로임을 고려한 비용을 포함한다. 비록 외적인 비용이 일반적으로 내적인 비용에 순수 판매수익금을 더한것이라고 하더라도 일부 경우에는 경제적인 요인과 심리적인 요인이 중요하다. 실례로 열심히 일할것을 요구하는 개발자들에게 의뢰자들은 내적인 비용보다 적은 비용을 보상하려고 할수 있다. 값이 설정된 기초우에서 계약이 맺어 질 때는 사정이 다르다. 의뢰자들은 결과적인 제품의 품질이 혹시 더 떨어 지게 될수 있는 경우를 고려하여 설정된 기타 다른 가격설정보다 더 낮게 가격설정하는것을 거절할수도 있다. 따라서 개발팀은 설정된 값을 약간 올리면서도 다른 경쟁자들이 설정했다고 보아 지는 가격보다는 좀더 낮게 값을 설정할수 있다.

계획에서 또 다른 하나의 중요한 부분은 프로젝트의 개발기간에 대한 타산이다. 의뢰자는 확실히 완성된 제품이 배포될 시기를 알고 싶어한다. 만약 개발팀이 그러한 일정계획을 가지고 있지 않으면 좋게는 기업체가 신용을 잃게 되며 나쁘게는 벌금이 적용된다. 모든 경우에 쏘프트웨어프로젝트관리계획에 책임이 있는 관리자는 해야 할것들에 대하여 많은 설명을 해야 한다. 반대로 개발기업체가 제품을 개발하는데 요구되는 시간을 과대평가하면 의뢰자들이 다른 기업체로 찾아 가게 된다.

유감스럽게도 비용이나 기간에 대한 요구를 정확히 타산하는것은 결코 쉽지 않다. 비용이나 기간을 정확히 타산하는데는 아주 많은 요인들이 있다. 중요한 요인의 하나는 인적요인이다. 30여년전에 쌔크맨(Sackman)과 그의 동업자들은 프로그람작성자들사이에 28에서 1까지의 차이가 있다는것을 발견하였다[Sackman, Erikaon, and Grant, 1968]. 경험 있는 프로그람작성자가 새로 시작한 프로그람작성자보다 항상 더 먼저 일을 수행한다고 말함으로써 그들의 결과를 쉽게 거절해 버릴수 있지만 쌔크맨과 그의 동료들은 조화가 맞는 여러 쌍의 프로그람작성자들을 비교하였다. 실례로 그들은 류사한 류형의 프로젝트에 대하여 10년간의 경험을 가진 두명의 프로그람작성자를 관찰하고 그들이 코드작성과 오유수정과 같은 과제를 수행하는 시간을 측정해 보았다. 그다음에는 짧은기간에 전문가수준에 오르고 류사한 교육학적인 배경을 가진 두명의 프로그람작성자들을 관찰해

보았다. 성능상 좋고 나쁨을 비교하여 제품의 크기에서는 6부터 1까지, 제품의 실행시간에서는 8에서 1까지, 개발시간에서는 9에서 1까지, 코드작성시간에서는 18에서 1까지, 오유수정시간에서는 28에서 1까지 차이가 있다는것을 관측하였다. 특히 놀라운것은 11년의 경험을 가진 두명의 프로그람작성자들에게서도 하나의 제품에 대한 성능상 좋고 나쁨이 있다는것이다. 지어 째크맨의 실례에서 좋고 나쁜 경우를 제쳐 놓고서도 관측된 차이는 5에서 1까지에 있었다. 이러한 결과들에 기초하여 명백히 비용과 시간을 임의의 정확도로 타산할수 있다는것은 기대할수 없다. 큰 프로젝트에 대하여서는 개별적인 사람들속에서의 차이가 무시되는 경향이 있다고 주장하는것도 있지만 이것은 아마도 우리가 바라던것이다. 즉 하나 또는 두개의 아주 훌륭한(또는 아주 나쁜) 개발팀성원들이 있다고 해도 일정계획에서 뚜렷한 차이가 있게 되고 예산에 대해서도 결정적인 영향을 준다.

타산에 영향을 줄수 있는 또 하나의 인적요인은 많은 나라들에서 프로젝트개발기간에 결정적인 역할을 하는 성원들이 사직하지 않을것이라는것을 담보하지 못한다는것이다. 그때 사직한 자리를 채우고 팀에서 교체를 진행하거나 혹은 남아 있는 성원들이 잃은것을 보상해야 한다는것을 인식하는데는 많은 시간과 비용이 든다. 두 경우에 다 일정계획을 잘못 세우고 타산이 잘못 진행되였다.

비용타산문제에 대하여서는 또 다른 문제가 있다. 즉 제품의 크기를 어떻게 측정하겠는가 하는것이다.

9. 2. 1. 제품의 크기에 대한 척도

제품의 크기에 대한 가장 일반적인 척도는 코드의 행수를 리용하여 계량하는것이다. 일반적으로 두개의 서로 다른 단위들이 리용된다. 즉 코드의 행들(LDC)과 배포된 천개의 원천명령(KDSI)을 리용한다. 많은 문제들이 코드의 행수와 관련되여 있다[van der Poel and Schach, 1983].

1. 원천코드를 작성하는것은 전체적인 쏘프트웨어개발노력의 적은 부분에 지나지 않는다. 원형을 작성하고 명세를 작성하며 계획작성, 설계, 실현, 통합, 시험하는데 요구되는 시간은 모두 마지막에 완성된 제품의 원천코드행수의 함수로 표현할수 있다고 하는것은 어딘가 다소 흥미 있는 문제로 될것 같다.
2. 같은 제품을 두개의 서로 다른 언어로 실현하면 코드의 행수가 서로 다르게 된다. 또한 Lisp와 같은 언어들이나 혹은 많은 비수속형4GL들을 리용하여 코드작성하면 코드의 행수에 관한 개념을 정의할수 없다.
3. 흔히 코드의 행수를 어떻게 계산하는가는 정확히 밝혀 지지 않는다. 다만 실행할수 있는 코드의 행만을 계산할수 있는가 혹은 자료의 정의도 계산할수 있는가? 그리고 설명문은 계산할수 있는가? 계산할수 없다면 프로그람작성자들이 《비생산적》이라고 여기는 설명문들에 시간을 랑비하는것이 혐오스럽다고 할수 있는데 이것은 위험하다. 그러나 설명문들이 계산되면 프로그람작성자들이 자기들의 생산성을 올리려는 시도에서 설명문들을 속여 가며 쓰는것도 상반되는 위험으로 된다. 또한 일감조종언어명령문들을 계산하는데 대해서는 어떻게 생각하는가? 또 하나의 문제는 성능을 개선하려고 제품을 보강하고 때로는 코드의 행수를 줄이는

과정에 변화된 행수나 삭제된 행수들을 어떻게 계산하는가 하는것이다. 코드의 재리용(8.1)은 또한 행의 계산을 포함하고 있다. 즉 만일 재리용된 코드가 수정되면 그것을 어떻게 계산하는가? 또한 만일 코드가 어미클라스로부터 계승되면(7.8) 어떻게 계산하는가? 간단히 말하여 정확하면서도 간단한 코드행수의 척도는 바로 무엇이나 다 간단히 세는것이다.

4. 작성된 모든 코드들이 의뢰자에게 배포되지는 않는다. 코드의 절반정도가 개발 노력을 지원하는데 필요한 도구들로 구성되여 있는것이 보통이다.
5. 쏘프트웨어개발자들이 보고서생성프로그람이나 화면생성프로그람 혹은 도형사용자대면부(GUI)생성프로그람과 같은 코드생성프로그람들을 리용한다고 가정하자. 개발자의 일부가 설계활동을 몇분정도 진행한 다음 도구들에 의하여 수천행되는 코드가 생성된다.
6. 일단 제품이 완전히 완성될 때만 완성된 제품에 들어 있는 코드의 행수를 결정할수 있다. 그러므로 코드의 행수에 기초하여 비용타산을 하는것은 의심할바없이 위험한것이다. 평가공정을 시작하자면 완성된 제품에 있는 코드의 행수를 평가하여야 한다. 그다음 이 평가결과는 제품의 비용을 계산하는데 리용한다. 모든 비용타산수법들에는 불확정성이 있을뿐만아니라 불확정적인 비용타산 그자체에 대한 입력자료가 불확실하면 즉 아직 채완성되지 않은 제품에 들어 있는 코드의 행수가 입력되면 그때에는 비용타산결과에 대한 믿음성이 높지 않을수 있다.

코드의 행수는 이처럼 믿을수 없기때문에 다른 척도를 고찰하여야 한다. 이른바 쏘프트웨어과학[Halstead, 1977]에서는 제품크기에 대한 여러가지 측정방법들을 제시하고 있다. 이것들은 쏘프트웨어과학의 기본적인 척도, 즉 쏘프트웨어제품에 있는 연산과 연산자의 수, 특정한 연산과 특정한 연산자의 수로부터 결과를 얻을수 있다. 코드의 행으로 계산하던 방법과 마찬가지로 제품이 일단 완성된 다음에만 이와 같은 계산을 할수 있으며 따라서 척도에서 예측능력이 크게 감소된다. 또한 많은 연구들은 쏘프트웨어과학의 타당성에 대하여 의심을 해소하여 주고 있다[Shepperd, 1988a; Weyuker, 1988b; Shepperd and Ince, 1994].

제품의 크기를 평가하는 또 다른 연구방법은 쏘프트웨어개발공정에서 신속하게 결정할수 있는 측정가능한 량들에 기초한 척도를 리용하는것이다. 실례로 반더포웰(van der Poel)과 스캐취(Schach)는 중간정도규모의 자료처리제품 즉 2～10명이 1년동안에 완성할수 있는 제품에 대한 비용타산을 진행하기 위하여 FFP척도를 제기하였다[van der Poel and Schach, 1983]. 자료처리제품에서 세가지 기초적인 구조적요소들은 그것의 파일(File), 흐름(Flow), 처리(Process)들이다. 즉 FFP라는 이름은 이러한 요소들의 첫 문자들로 이루어 진 략어이다. 파일은 제품에 항구적으로 들어 있는 물리적으로 또는 물리적으로 련관된 기록들의 집합으로서 정의된다. 즉 트랜잭션과 림시적인 파일들은 제외된다. 흐름은 화면이나 보고서와 같이 제품과 주위환경사이의 자료대면부이다. 처리는 자료들에 대한 론리적인 또는 산수적인 조작으로 정의된다. 즉 실례로 처리는 정돈이나 타당성검증, 갱신들을 포함하고 있다. 파일의 수는 Fi, 흐름의 수는 Fl, 처리의 수는 Pr라고 하고 제품에서 그 크기를 S, 비용은 C라고 하면 다음과 같이 계산할수 있다. 즉

$$S = Fi + Fl + Pr \tag{9.1}$$

$$C = d \times S \tag{9.2}$$

여기서 d는 기업체들에 따라서 변화되는 상수이다. 상수 d는 해당한 기업체안에서 쏘프트웨어개발공정의 효과성(생산성)에 대한 측정값이다. 제품의 크기는 간단히 파일과 흐름, 처리의 개수들의 합으로 되며 일단 구성방식설계가 완성되면 결정될수 있는 량이다. 비용은 제품의 크기에 비례하며 비례상수 d는 해당 기업체에서 이미 개발한 제품과 관련한 비용자료에 적합한 최소두제곱에 의하여 결정된다. 코드의 행수에 기초한 척도와는 달리 비용은 코드작성시작전에 타산될수 있다.

FFP척도의 타당성과 믿음성은 중간규모의 자료처리응용프로그람의 범위내에 있는 목적하는 실례프로그람을 리용하여 보여 줄수 있다. 유감스럽게도 척도는 자료기지 즉 많은 자료처리제품의 본질적인 구성요소들에는 절대로 확장되지 못한다. 독립적으로 개발한 제품의 크기에 대한 척도는 기능점들에 기초하여 엘브레취트(Albrecht)가 개발하였다[Albrecht, 1979]. 엘브레취트의 척도는 입력항목들의 수 *Inp*, 출력항목들의 수 *Out*, 요구수 *Inq*, 주파일의 수 *Maf*, 대면부의 수 *Inf*에 기초하고 있다. 가장 간단한 방법으로서 기능점들의 수 FP는 다음의 식

$$FP = 4 \times Inp + 5 \times Out + 4 \times Inq + 10 \times Maf + 7 \times Inf \tag{9.3}$$

에 의해서 계산된다. 이것은 제품의 크기에 대한 척도이기때문에 비용타산과 생산성평가에 리용될수 있다.

식 (9.3)은 세단계의 계산으로 간단히 진행된다. 첫째로, 조절되지 않은 기능점들이 계산된다.

1. 제품의 매개 구성요소들 *Inp*, *Out*, *Inq*, *Maf*, *Inf* 들은 단순한것, 보통인것, 복잡한것으로 분류되여야 한다(그림 9-2를 보시오).

	복잡성의 준위		
구성요소	**간단한 상태**	**중간상태**	**복잡한 상태**
입력항목	3	4	6
출력항목	4	5	7
요구	3	4	6
주파일	7	10	15
대면부	5	7	10

그림 9-2. 기능점값표

2. 매개 구성요소들은 그 준위에 따라 많은 기능점들에 부가된다. 실례로 보통의 입력값은 식 (9.3)에 반영된바와 같이 네개의 기능점들에 부여된다. 그러나 단순한 입력값은 오직 세개이고 반면에 복잡한 입력값은 6개의 기능점들에 부가된다. 이 단계에서 요구되는 자료들은 그림 9-2에 보여 준다.
3. 매개 구성요소들로 부여된 기능점들은 그다음 합하여 조절이 되지 않은 기능점(*UFP*)을 부여한다.

둘째로, 기술적인 복잡성인자(TCF)가 계산된다.

1	자료통신
2	분산자료처리
3	성능척도
4	중요한 하드웨어
5	높은 트랜잭션비률
6	직결자료입력점
7	말단사용자의 효과성
8	직결갱신
9	복잡한 계산
10	재리용가능성
11	쉬운 설치
12	쉬운 조작
13	이식가능성
14	유지정비성

그림 9-3. 기능점계산의 기술적요인

이것은 14개의 기술적인 요인들의 효과에 대한 측정이다. 기술적인 요인들에는 높은 트랜잭션비률, 성능기준(실례로 처리능력 혹은 응답시간), 직결갱신 등이 있으며 완전한 요인들을 그림 9-3에 보여 준다. 이러한 매 14개의 요인들은 0(《없거나 영향이 없다.》)부터 5(《전반에 걸쳐 강하게 영향을 준다.》)까지에서 값을 부여한다. 결과 14개의 수를 합하여 전체적인 영향정도(DI)를 얻는다. TCF는 그다음 다음의 식에 의하여 계산된다. 즉

$$TCF = 0.65 + 0.01 \times DI \tag{9.4}$$

DI가 0～70까지 변화될수 있기때문에 TCF는 0.65～1.35사이에서 변할수 있다.

셋째로, 기능점의 수 FP는 다음과 같이 계산된다.

$$FP = UFP \times TCF \tag{9.5}$$

쏘프트웨어생산능률을 계산하기 위한 실험을 진행한데 의하면 KDSI를 리용하는것보다 기능점들을 리용하는것이 더 좋다는것을 보여 주었다. 실례로 죠네스(Jones)는 KDSI를 계산하면서 800%를 초과하는 결함을 발견하였지만 기능점을 계산할 때에는 겨우 200%만을 발견하였다고 지적하였다[Jones, 1997].

이 결과는 매우 훌륭하다.

코드행에 비한 기능점의 우월성을 보여 주기 위하여 죠네스는 그림 9-4에 실례를 보여 주었다[Jones, 1997]. 같은 제품을 아쎔블리어와 Ada로 코드작성하여 결과를 비교하였다. 우선 월공수당 KDSI를 고찰하여 보자. 이 척도는 아쎔블리어에서의 코드작성이 얼

핏 보기에는 Ada에서의 코드작성보다 60% 더 효률적이라는것을 보여 준것 같지만 이것은 명백히 잘못된 결론이다. Ada와 같은 3세대언어들은 단지 3세대언어로 코드작성하는것이 훨씬 더 효과적이라는 사정으로 하여 아쎔블리어대신에 리용하고 있다. 이번에는 두번째 척도인 원천명령문당 비용을 고찰하여 보자. 이 제품에서 하나의 Ada명령문은 2.8개의 아쎔블리어명령과 동등하다는것을 생각해 보시오. 효과성의 크기로서 원천명령문당 비용을 리용하면 Ada보다 아쎔블리어로 코드작성하는것이 더 효과적이라는것을 알수 있다. 그러나 월공수당 기능점을 프로그람작성효과성의 척도로 취하는 경우 아쎔블리어에 비한 Ada의 우월성이 명백히 알려 진다.

	아쎔블리어판본	**Ada판본**
원천코드크기	70KDSI	25KDSI
개발비용	1,043,000딸라	590,000딸라
월공수당KDSI	0.335	0.211
원천명령당 비용	14.90딸라	23.60딸라
월공수당 기능점	1.65	2.92
기능점당 비용	3,023딸라	1,170딸라

그림 9-4. 아쎔블리어제품과 Ada제품의 비교[Jones, 1987].[©1987 IEEE.]

다른 한편 기능점뿐만아니라 식 (9.1)과 (9.2)의 FFP척도는 이와 같은 약점을 가지고 있다. 즉 제품의 유지정비가 종종 부정확하게 계산된다. 하나의 제품을 유지정비하자면 그 제품에 큰 변화를 줄 때 파일, 흐름, 처리의 수 또는 입력, 출력, 요구, 주파일, 대면부의 수가 변화되지 말아야 한다. 이러한 견지에서 코드의 행들을 고려하는것은 더 좋은것이 아니다. 최악인 경우에 매개 행들을 코드행의 총 수를 변화시키지 않는 상태에서 완전히 다른 행으로 교체하는것이 가능하다.

엘브레취트의 기능점에 대한 확장으로서 최소한 40개의 변종이 제기되였다[Maxwell and Forselius, 2000]. 조절되지 않은 기능점(*unadjusted function points*, UFP)을 계산하기 위한 보다 좋은 방법을 찾기 위하여 씨몬스(Symons)는 MKⅡ기능점을 제시하였다[Symons, 1991]. 쏘프트웨어를 매개가 입력, 처리, 출력으로 이루어 진 구성요소트랜잭션모임으로 분해한다. 그다음 입력, 처리, 출력으로부터 *UFP*값을 계산한다. MKⅡ기능점들은 세계적으로 널리 리용되고 있다[Boehm, 1997].

9. 2. 2. 비용타산방법

크기를 평가하는데 난점이 있음에도 불구하고 개발자들이 평가에 영향을 줄수 있는 많은 요인들을 가능한껏 고려하면서 프로젝트의 기간과 프로젝트의 비용을 정확히 계산하기 위하여 할수 있는 모든것을 다 하는것이 중요하다. 평가에 영향을 줄수 있는 요인들에는 인간의 기능수준, 제품의 복잡성, 제품의 크기(비용은 크기에 따라 증가한다.), 응용분야에 대한 개발팀의 파악, 그 제품에 적용될 하드웨어, CASE도구를 쓸수 있는가 하는것들이 포함된다. 다른 하나의 요인은 이른바 최종기한이 주는 영향이다. 만일 제품

이 어느 때까지 완성되여야 한다면 월공수는 마감기한이 주어 지지 않을 때보다 더 크다.

모든 정황을 다 포괄했다고는 볼수 없는 우의 목록만 보아도 평가가 힘든 문제이라는것을 명백하게 알수 있다. 여러가지 방법을 적용하여 보았는데 성공여부는 각이하다.

1. 류추에 의한 전문가평가 이 기법에서는 많은 전문가들을 객체로 하고 있다. 전문가는 목적하는 제품과 자기가 적극 참여하여 완성한 제품을 비교하여 류사한 점과 차이점을 알아 내는 방법으로 평가를 내릴수 있다. 실례로 전문가는 직결자료식별능력을 가져야 할 목적하는 제품을 묶음방식으로 자료를 넣어 주는 2년전에 개발한 류사한 제품과 비교할수 있다. 개발팀이 개발하여야 할 제품의 류형을 잘 알고 있기때문에 전문가는 개발기한과 노력을 15%줄일수 있었다. 그러나 도형사용자대면부(GUI)는 아주 복잡하다. 즉 이것은 시간과 노력을 25%나 증대시킨다. 종합적으로 말하면 목적하는 제품은 대부분이 개발팀성원들이 잘 알지 못하는 언어로 개발하는 경우 기간은 15%, 노력은 25% 늘어 난다. 이 세가지 수자를 종합하면서 전문가는 목적하는 제품이 이전의것보다는 시간이 25%, 노력은 30% 더 들것이라고 판단하게 된다. 이전의 제품이 완성되는데 12개월의 시간과 100명이 1개월동안 진행하는 로력이 든다면 목적하는 제품은 15개월의 시간과 130명이 1개월동안 진행하는 로력이 요구될것이다.

그 기업체에 있는 다른 두 전문가는 같은 두 제품을 비교하였다.

한 사람은 최종제품개발에 13.5개월의 시간과 140명이 1개월동안 진행하는 로력이 들것이라고 보았다. 다른 사람은 16개월의 시간과 95명이 1개월동안 진행하는 로력이 든다고 보았다. 이 세 사람의 예측을 어떻게 조정하겠는가? 한가지 방법은 델피(Delphi)기법이다. 이 기법은 기업체협의회를 가지지 않고도 결론에 도달할수 있게 해준다. 그것은 집단을 움직일수 있는 설득력을 가진 사람이 한명이라도 있으면 좋지 않은 부작용을 초래할수 있다. 이 기술을 도입할 때 전문가들은 독자적으로 사업한다. 각자는 자기 식의 평가를 진행하고 그 평가에 대한 근거를 밝힌다. 그다음 이러한 평가와 근거들을 전문가들에게 배포한다. 그러면 이번에는 모든 전문가들이 두번째 평가를 진행한다. 전문가들이 하나의 공통적인 평가를 내릴 때까지 이러한 평가와 배포작업을 계속 진행한다. 이러한 반복과정기간에는 기업체협의회를 진행하지 않는다.

실지 부동산의 가치는 류추를 리용한 전문가적평가에 기초하여 결정한다. 평가자는 어느 한 집을 최근에 팔린 류사한 집과 비교하는 방법으로 그 가치를 결정하게 된다. A라는 집의 값을 결정하려고 하는데 B라는 옆집은 205,000딸라에 금방 팔리웠고 이웃거리에 있는 집 C는 석달전에 218,000딸라에 팔리웠다고 가정하자.

이런 경우에 평가자는 다음과 같이 추리할수 있다. 즉 집 A는 집 B보다 침실이 하나 더 많고 마당은 5,000평방피트 더 크다. 집 C는 집 A와 크기가 거의 비슷하지만 지붕상태는 한심하다. 한편 집 C에는 목욕탕용분사식물흐름장치가 있다. 주의 깊게 타산해 본 평가자는 집 A의 값을 215,000딸라로 결정할수 있다.

쏘프트웨어제품인 경우 류추를 리용한 전문가평가는 실지 부동산가치타산보다 정확하지 못하다. 첫번째로 쏘프트웨어전문가가 파악이 없는 언어를 리용하는 경우 시간은 15%, 노력은 20%로 늘어 날것이라는데 대하여 상기해 보아야 한다. 전문가에게 매개의 차이로 하여 초래할수 있는 영향을 예견할수 있게 하는 실용성 있는 자료가 주어지지 않은 경우에는 추측이라고밖에 달리는 생각할수 없는 그러한 오유로 하여 정확

하지 못한 비용타산을 하게 된다. 게다가 전문가가 모든것을 기억해 낼수 없다면(또는 상세한 기록을 해놓치 않았다면) 완성된 제품들에 대한 기억은 부정확할수밖에 없다. 총체적으로 전문가는 인간이므로 정확한 예측을 할수 없다는 견해도 있다. 동시에 한 전문가기업체에 의한 평가결과는 그들의 집체적인 경험을 반영하는것으로 된다. 즉 만일 이러한 경험이 실지로 포괄적인것이라고 하면 결과는 정확한것으로 된다.

2. 상승식방법 어떤 제품자체의 가격을 타산하는데서 발생하는 오유를 극복하는 한가지 방도는 그 제품을 보다 작은 구성부분들로 분할하는것이다. 그리고 그 매 구성부분들의 개발기간과 비용을 계산하고 그 값을 합하는 방법으로 전체 가격을 계산한다. 이 방법은 보다 작은 구성부분들에 대한 비용타산이 일반적으로 전체에 대하여 진행하는 계산보다 더 빠르고 정확하다는 우점을 가지고 있다. 또한 평가과정이 보다 더 세부적이라는것이다. 이 방법의 약점은 제품값이 그 구성부분들의 값들을 합한것보다 크다는것이다.

객체지향파라다임과 관련하여 여러가지 클라스들의 독립성은 상승식방법을 도와 주고 있다. 그러나 제품에 있는 여러가지 객체들속에서 있게 되는 호상작용은 평가과정을 복잡하게 한다.

3. 알고리듬적인 비용타산모형 이 방법에서는 기능점이나 FFP와 같은 척도가 제품의 비용을 결정하기 위한 어느 한 모형에 입력자료로 리용된다. 타산자는 척도값을 계산한다. 즉 기간과 비용타산은 그 모형을 리용하여 계산된다. 표면상 알고리듬적인 비용타산모형은 전문가들의 의견에 비하여 우월하다. 왜냐하면 전문가는 우에서도 언급한바와 같이 편차가 있으며 완성된 제품과 목적하는 제품의 일부 측면들을 스쳐 지나갈수 있기때문이다. 이와 달리 알고리듬적인 비용타산모형은 편차가 없다. 즉 매 제품이 같은 방법으로 처리된다. 이 모형을 리용할 때의 위험은 그 타산이 기초적인 추측에 불과하다는것이다. 실례로 기능점모형이 기초로 하고 있는것은 제품의 매 측면이 식 (9.3)의 오른쪽에 있는 다섯개의 량들과 14개의 기술적요인들에 구현되여 있다고 하는 추측이라는것이다. 그다음의 문제점은 모형의 파라메터들에 어떤 값을 부여 하겠는가 하는데 주관적인 판단이 중요하게 요구된다는것이다. 실례로 기능점모형의 특정한 기술적인 요인이 3에 비례하겠는지 아니면 4에 비례하겠는지 명백하지 않을 때가 많다.

알고리듬적인 비용타산모형들은 많이 제기되였다. 어떤 모형들은 쏘프트웨어가 어떻게 개발되는가 하는것과 관련한 수학적리론들에 기초하고 있다. 다른 모형들은 통계학적인 리론에 기초하고 있다. 즉 많은 프로젝트들이 연구되고 그러한 자료로부터 경험적인 규칙들이 확립된다. 혼성모형은 수학적인 식들과 통계적인 모형 그리고 전문가적인 판단을 통합하고 있다. 가장 중요한 혼성모형은 보엠(Boehm)의 COCOMO로서 이에 대하여서는 다음절에서 상세히 설명한다(준말 COCOMO에 대하여서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

COCOMO는 COnstructive COst MOdel에서 매 단어의 첫 두 글자를 따서 만든 준말이다. 인디아의 KoKomo와 그 어떤 관련이 있는 소리 같은 말이다. *COCOMO*모형이i ·는 표현은 쓰지 말아야 한다. 즉 COCOMO에서 《*MO*》는 《모형》을 의미한다.

9. 2. 3. 중간 COCOMO

COCOMO는 실제상 전체적인 제품을 취급하는 거시적인 타산모형으로부터 제품을 세부적으로 취급하는 미시적인 타산모형에 이르기까지 세가지 모형들의 련속이다. 이 절에서는 중간 COCOMO에 대한 설명을 주고 있으며 이것은 복잡한것과 세부적인것에 대하여 중간준위를 가진다는것을 의미한다. COCOMO는 문헌 [Boehm, 1981]에서 상세히 서술하였다. 즉 문헌 [Bochm, 1984b]에서는 그것에 대한 개관을 주었다.

중간 COCOMO를 리용하여 개발기간을 타산하는것은 두 단계로 진행한다. 첫째로, 개발노력에 대한 대략적인 평가가 주어 진다. 여기서는 두개의 파라메터들이 평가되여야 한다. KDSI로 계산된 제품의 크기와 제품개발방식 그리고 그 제품들을 개발하는 난관의 내부준위에 대한 측정이 평가되여야 한다. 여기에는 세가지 방식 즉 작고 간단한 단순한 방식(*organic*), 크기가 중간정도인 중간방식(*semidetached*) 복잡한 방식(*embedded, complex*)이 있다.

이 두 파라메터들로부터 명목상의 로력(*nominal effort*)을 계산할수 있다. 실례로 만일 프로젝트가 본질에 있어서 단순한것으로 판명되면 명목상 로력은 다음의 식으로 계산된다. 즉

$$\text{명목상 로력} = 3.2 \times (\text{KDSI})^{1.05} \ \text{월공수} \tag{9.6}$$

3.2와 1.05는 중간 COCOMO를 개발하기 위하여 보엠이 리용한 단순한 방식의 제품에 대한 자료에 가장 적합한 값들이다. 실례로 만일 개발될 제품이 단순하고 12,000개의 원천명령문들(12KDSI)로 평가된다면 명목상 로력은 다음과 같이 계산된다. 즉

$$3.2 \times (12)^{1.05} = \text{월공수}$$

(그러나 이 값에 대한 해설을 보려면 다음의 《알고 싶은 문제》를 보시오).

알고 싶은 문제

명목상 로력의 값에 대한 한가지 반응은 《만일 매 43명의 사람들이 1개월동안 진행하는 로력(월공수)이 12,000개의 원천명령문들을 작성하는데 요구되였다면 평균 매 프로그람작성자는 1개월동안에 300개 행보다 더 적은 코드를 작성한것으로 된다. 》

300행되는 제품은 바로 코드의 행수가 300행이라는것을 의미한다. 대비적으로 유지정비가능한 12,000행되는 제품은 생명주기의 모든 단계들을 다 거쳐야 한다. 다시 말하여 43명이 1개월동안 진행하는 전체 로력은 코드작성을 비롯하여 많은 작업들을 포함하게 된다. 그림 1-2의 파라다임도표에서 보여 준것처럼 모듈의 코드작성은 평균 총 개발로력의 15%정도밖에 되지 않는다.

다음으로 이러한 명목상 로력은 15개의 쏘프트웨어프로젝트 개발로력승수 (*software effort multiplier*) 들을 곱해야 한다. 이러한 승수들과 그 값들을 그림 9-5에 보여 주었다. 매 승수는 6개까지의 값을 가진다. 실례로 제품복잡성승수에는 개발자들이 프로젝트의 복잡성을 매우 낮은것, 약간 낮은것, 보통인것, 좀 높은것, 매우 높은것 혹은 극도로 높

은것으로 평가하는데 따라서 그 값들은 각각 0.70, 0.85, 1.00, 1.15, 1.30, 1.65로 할당하고 있다. 그림 9-5에서 볼수 있는것처럼 15개의 모든 승수들은 해당한 파라메터가 보통일 때 값 1.00을 가진다.

비용결정요인	등급					
	아주 낮다	낮다	보통	높다	아주 높다	극도로 높다
제품특성						
요구하는 쏘프트웨어믿음성	0.75	0.88	1.00	1.15	1.40	
자료기지크기		0.94	1.00	1.08	1.16	
제품의 복잡성	0.70		1.00	1.15	1.30	1.65
콤퓨터특성						
실행시간제한			1.00	1.11	1.30	1.66
기본보관제한			1.00	1.06	1.21	1.56
가상기계휘발성*		0.87	1.00	1.15	1.30	
콤퓨터순회시간		0.87	1.00	1.07	1.15	
성원특성						
분석자능력	1.46	1.19	1.00	0.86	0.71	
응용경험	1.29	1.13	1.00	0.91	0.82	
프로그람작성자능력	1.42	1.17	1.00	0.86	0.70	
가상기계경험*	1.21	1.10	1.00	0.90		
프로그람작성언어경험	1.14	1.07	1.00	0.95		
프로젝트특성						
현대프로그람작성실천에서의 리용	1.24	1.10	1.00	0.91	0.82	
쏘프트웨어도구	1.24	1.10	1.00	0.91	0.83	
요구되는 개발일정계획	1.23	1.08	1.00	1.04	1.10	

* 주어 진 쏘프트웨어제품에 대하여 기초적인 가상기계는 과제를 수행하기 위하여 요구하는 하드웨어와 쏘프트웨어(조작체계, 자료기지관리체계)의 복합체이다.

그림 9-5. 중간 COCOMO 쏘프트웨어개발로력승수

［Boehm, 1984b］. ［©1984 IEEE.］

보엠은 파라메터가 실제적으로 비률이 명목상인가 혹은 그보다 더 낮은가 혹은 더 높은가 하는것을 개발자가 확증하는데 도움을 줄수 있는 지도서를 내놓았다. 실례로 모듈복잡성승수를 다시 생각해 보자. 만일 모듈의 조종연산들이 본질상(**if-then-else**, **do-while**, **case**와 같은) 구조화프로그람작성의 구성들의 순차적인 렬로 이루어 져 있다면 복잡성은 대단히 낮은것으로 평가된다. 만일 이 연산자들이 함유되여 있으면 비률은 낮다. 내부모듈의 조종과 결정표들을 추가하는것은 명목상의 평가를 증가시키는것으로 된다. 만일 연산자들이 복합술어들로 고도로 함유되여 있고 대기렬들과 탄창들이 있다면 평가는 높게 설정된다. 재입력가능하고 재귀적인 코드작성과 고정된 우선권중단처리가

있으면 평가를 대단히 높은것으로 되게 한다. 끝으로 동적으로 변화하는 우선권들과 마이크로코드준위조종을 가지는 다중자원처리계획작성은 평가를 극도로 높은것으로 설정한다. 이러한 비률은 조종연산들에도 적용된다. 모듈은 또한 계산처리와 장치의존적인 처리, 자료관리연산의 견지에서도 평가되여야 한다. 매 15개의 승수를 계산하기 위한 기준에 대하여 자세한 내용을 알려면 문헌 [Boehm, 1981]을 참고하시오.

이것을 어떻게 하는가를 보기 위하여 문헌 [Boehm, 1984b]에서는 고도로 믿음성이 있고 새로운 전자식자금전송망을 위한 극소형처리소자에 기초한 자료통신처리쏘프트웨어의 실례를 성능과 개발일정계획, 대면부요구사항을 포함하여 제시하고 있다. 이 제품은 내장방식의 서술에 적합하며 크기는 1만개의 원천명령문(10KDSI)으로써 평가되였다. 그리하여 명목상 개발로력은

$$\text{명목상 로력} = 2.8 \times (\text{KDSI})^{1.20} \tag{9.7}$$

으로 계산된다(여기서 상수 2.8과 2.0은 내장된 제품에 대한 자료에 가장 적합한 값들이다.). 프로젝트가 크기상 10KDSI로 평가되기때문에 명목상 로력은

$$2.8 \times (10)^{1.20} = 44\text{월공수}$$

으로 평가된다. 평가된 개발로력은 명목상 로력을 15개의 승수들을 곱하여 얻어 진다. 이 승수들과 그 비률값들을 그림 9-6에 주었다. 이 값들을 리용할 때 승수값은 1.35이며 따라서 이 프로젝트를 위하여 평가된 로력은

$$1.35 \times 44 = 59\text{월공수}$$

로 된다.

이때 이 수는 딸라비용, 개발일정계획, 단계와 활동의 분포, 콤퓨터비용, 년간유지정비비용, 기타 다른 년간항목들을 확정하기 위한 추가적인 공식으로 리용된다. 자세한 것은 문헌 [Boehm, 1981]에 있다. 중간 COCOMO는 완전한 알고리듬적인 비용타산모형으로서 사용자들에게 프로젝트계획작성에서 실제적으로 있을수 있는 방조들을 모두 제공하여 준다.

중간 COCOMO는 방대하고 다양한 응용분야들을 포함하는 63개의 프로젝트들에 대한 일반적인 실례들과 관련하여 타당하게 되였다. 중간 COCOMO를 이러한 실례들에 적용한 결과 실제값이 대략 시간의 68%이라고 하는 예측값의 20%이내에 있다는것이다. 중간 COCOMO를 위한 입력자료가 불과 ±20%정도내까지 정확하기때문에 대부분의 기업체들에서 이 정확도를 높이기 위한 시도는 거의나 없었다. 그럼에도 불구하고 경험 있는 평가자들에 의하여 얻어 진 정확성은 중간 COCOMO를 1980년대기간에 진행한 비용타산연구의 뚜렷한 효과로 인정하게 하였다. 즉 그 어떤 다른 기법도 일관하게 정확성을 보장하지 못하였다.

중간 COCOMO에서 중요한 문제는 그것에 대한 가장 중요한 입력이 목적하는 제품에 있는 코드의 행수이라는것이다. 만일 이러한 평가가 정확하지 않으면 그때에는 모형에 대한 모든 단순한 예측이 부정확하게 될수 있다. 중간 COCOMO 예측 또는 그 어떤 다른 평가방법이 부정확할수도 있다는 가능성이 있는것으로 하여 관리자측은 쏘프트웨어 개발전반에 걸쳐 모든 예측들을 감시하여야 한다.

비용결정요인	상　　황	등급	로력승수
요구되는 쏘프트웨어믿음성	쏘프트웨어고장의 엄중한 재정적인 후과	높다	1.15
자료기지크기	20,000바이트	낮다	0.94
제품의 복잡성	통신처리	아주 높다	1.30
실행시간제한	리용가능한 시간의 70% 리용	높다	1.11
기본보관제한	64K보관능력중에서 45K 보관(70%)	높다	1.06
가상기계휘발성	상업적인 마이크로처리기 하드웨어에 기초	보통	1.00
콤퓨터순회시간	두 시간 평균순회시간	보통	1.00
분석자능력	고급한 분석자작성자	높다	0.86
응용경험	3년	보통	1.00
프로그람작성자능력	고급한 프로그람작성자	높다	0.86
가상기계경험	6개월	낮다	1.10
프로그람작성언어경험	12개월	보통	1.00
현대프로그람작성실천의 리용	년간리용에서 최고기술	높다	0.91
쏘프트웨어도구	기초적인 소형콤퓨터도구준위	낮다	1.10
요구되는 개발일정계획	9개월	보통	1.00

그림 9-6. 극소형처리기통신쏘프트웨어를 위한 중간 COCOMO 쏘프트웨어개발로력승수［Boehm, 1984b］.［©1984 IEEE.］

9. 2. 4. COCOMO Ⅱ

COCOMO는 1981년에 제안되였다. 그때 사용하고 있던 유일한 생명주기모형은 폭포모형이였다. 의뢰자-봉사기와 객체지향과 같은 기법들은 본질에 있어서 알려 져 있지 않았다. 따라서 COCOMO는 이러한 요인들가운데서 그 어느 요인도 병합하지 않고 있다. 그러나 보다 더 새로운 기술들이 공인된 쏘프트웨어공학실천으로 등장하기 시작하자 COCOMO는 그 정확성이 낮아 지기 시작하였다.

COCOMO Ⅱ[Boehm et al., 2000]은 1981년의 COCOMO를 개정한 주요판본이다. COCOMO Ⅱ은 객체지향과 3장에서 설명한 여러가지 생명주기의 모형들, 신속원형작성(10.3), 4세대의 언어들(14.2), 재리용(8.1), COTS쏘프트웨어(2.1)를 비롯하여 아주 다양한 현대쏘프트웨어공학수법들을 처리할수 있다. COCOMO Ⅱ는 유연하고 정교하다. 유감스럽게도 이 목적을 달성하기 위하여 COCOMO Ⅱ는 또한 원래의 COCOMO보다 매우 복잡하다. 따라서 COCOMO Ⅱ를 리용하려고 하는 독자들은 문헌 [Boehm et al., 2000]을 자세히 연구하여야 한다. 즉 여기서는 다만 COCOMO Ⅱ와 중간 COCOMO사이의 중요

한 차이들에 대하여 개괄하고 있다.

우선 중간 COCOMO는 코드의 행(KDSI)에 기초한 한개의 총체적인 모형으로 이루어져 있다. 한편 COCOMO Ⅱ는 세개의 각이한 모형들로 이루어 져 있다. 객체점(기능점과 류사하다.)에 기초한 응용프로그람합성모형(*application composition model*)은 개발되는 제품과 관련하여 최소한의 지식이 리용되던 초시기의 단계들에서 적용된다. 그다음 더 많은 지식이 리용되게 되자 초기설계모형(*early design model*)이 리용된다. 즉 이 모형은 기능점들에 기초하고 있다. 마지막에는 개발자들이 최대의 정보를 가지게 되자 우편구성방식모형(*postarchitecture model*) 이 리용된다. 이 모형은 기능점들 혹은 코드행(KDSI)들을 리용한다. 중간 COCOMO로부터 나오는 출력은 비용과 크기의 타산이다.

즉 COCOMO Ⅱ의 세개 모형가운데서 매 모형에서 나오는 출력은 비용과 크기타산의 범위로 된다. 따라서 만일 로력에 대한 타산을 E라고 하면 응용프로그람합성모형은 범위(0.80E, 1.25E)를 귀환한다. 이것은 COCOMO Ⅱ의 모형들의 발전과정에 정확도가 높아 지는 과정을 반영하여 주고 있다.

두번째 차이는 COCOMO에 기초한 로력모형에 있다. 즉

$$로력 = a \times (크기)^b \tag{9.8}$$

이다. 여기서 a와 b는 상수이다. 중간 COCOMO에서는 지수 b에 대하여 세개의 서로 다른 값들을 주고 있는데 그 값들은 개발되는 제품의 방식이 단순한 방식(b=1.05)인지 중간방식(b=1.12)인지 혹은 복잡한 방식(b=1.20)인가에 달려 있다. COCOMO Ⅱ에서 b의 값은 1.01과 1.26사이에서 변하며 이것은 모형의 다양한 파라메터에 의존하고 있다. 이것들은 그러한 류형의 제품들과의 친숙성, 처리성숙준위(2.11), 위험해소의 크기(3.6절) 그리고 팀의 협동정도(4.1)를 포함하고 있다.

세번째 차이는 재리용과 관련한 추측이다. 중간COCOMO는 재리용으로 인한 절약이 재리용량에 직접적으로 비례하고 있다고 가정하고 있다. COCOMO Ⅱ는 재리용된 쏘프트웨어에서 발생한 자그마한 변화들이 불균형적으로 큰 변화들을 초래하고 있다고 생각하고 있다(왜냐하면 코드가 거의 자그마한 변화에 대하여서는 구체적으로 리해되며 수정된 모듈들을 시험하는 비용이 상대적으로 크기때문이다).

넷째로 중간 COCOMO안에는 15개대신 17개의 비용타산요인승수들이 있다. 비용타산요인중 7개는 앞으로의 제품들에서 요구될 재리용성과 매해 직원들의 재편성 그리고 제품이 여러곳에서 개발되고 있는가 하는것과 같은 새로운것들이다. COCOMO Ⅱ는 다양한 서로 다른 령역들에서 취한 83개의 프로젝트들을 리용하여 조사하였다. 모형은 여전히 너무 새로운것이여서 그것의 정확도와 특히 그것의 이전의 모형 즉 원래(1981년)의 COCOMO에 비하여 개선된 정도와 관련하여 많은 결과들이 나왔다.

9. 2. 5. 기간과 비용타산의 추적

제품이 개발되고 있는동안 실제적인 개발노력은 예측들과 끊임없이 비교되여야 한다. 실례로 쏘프트웨어개발자들이 리용한 타산량이 명세작성단계가 3개월 걸리고 7명이 1개월동안 하는 노력을 요구한다고 가정하자. 그러나 4개월이 지나 가고 10명이 1개월동안

하는 노력이 지출되였다. 그러나 아직 명세서는 결코 완전하지 못하다. 이러한 편향은 무엇인가 잘못된것이며 따라서 수정해야 한다는것을 초기에 경고할수 있다. 문제는 제품의 크기가 과소평가되였고 개발팀이 생각했던것만큼 충분한 능력이 없다는데 있을수 있다. 리유가 무엇이든간에 기간이 오래 지속되고 비용이 초과되였으므로 관리자는 그 영향을 최소화하기 위하여 적절한 조치를 취해야 한다.

예측에 대한 세밀한 추적은 예측을 진행한 기법들에 관계없이 개발전과정에 걸쳐 진행되여야 한다. 편향은 서투른 사용자와 불충분한 쏘프트웨어개발 그리고 량자의 결합과 기타 다른 리유로 되는 척도에 기인될수 있다. 중요한것은 편향들을 신속하게 찾아내고 즉시에 수정대책을 세우는것이다.

9. 3. 쏘프트웨어프로젝트관리계획의 구성요소

쏘프트웨어프로젝트관리계획에는 세개의 기본구성요소들이 있다. 즉 하여야 할 일, 그것을 하는데 필요한 자원들, 그 모든것에 지불해야 할 비용이 있다. 이 절에서는 이 세 구성요소들에 대하여 론의한다. 용어는 문헌 [IEEE 1058.1, 1987]에서 찾아 볼수 있는데 그것은 9.4에서 더 자세히 론의하기로 한다.

쏘프트웨어개발은 자원(*resource*)을 요구한다. 필요한 주요자원들로는 쏘프트웨어를 개발하는 사람들과 쏘프트웨어를 실행시킬 하드웨어, 조작체계, 본문편집프로그람, 판본조종쏘프트웨어(5.7)와 같은 지원쏘프트웨어들이다.

개발성원들과 같은 자원비용은 시기에 따라 달라 진다. 노덴(Norden)은 큰 프로젝트에 대하여 레일리(Rayleigh)분포가 자원소비 R_c는 시간 t에 따라 변화한다는것을 가장 가깝게 반영하고 있다는것을 문헌 [Norden, 1958]에서 보여 주었다. 즉

$$R_c = \frac{t}{k^2} e^{-t^2/2k^2} \quad 0 \le t < \infty \tag{9.9}$$

파라메터 k는 소모가 정점에 이른 시간 즉 상수이며 e=2.71828⋯는 나뽈레옹로그의 밑수이다. 전형적인 레이레그곡선을 그림 9-7에 보여 준다. 자원소모는 작게 시작하여 급격히 정점으로 뛰여 오르며 그다음에는 보다 낮은 속도로 줄어 든다. 푸트남(Putnam)은 쏘프트웨어개발에 대한 노덴의 결과들의 적용가능성을 조사하고 개발성원들과 다른 자원소모가 레이레그분포에 따라 일정한 정도로 정확성을 가지고 모형화되였다는것을 발견하였다[Putnam, 1978].

따라서 단순히 적어도 5년간의 경험을 가진 세명의 고급프로그람전문가들이 요구된다고 말하는것은 쏘프트웨어계획에서는 불충분하다. 다음과 같은 것들이 필요하다.

> 실시간 프로그람작성에서 최소한 5년간의 경험을 가진 3명의 고급프로그람전문가들이 필요하다. 2명은 프로젝트개발이 시작된 다음 3개월만에 개발에 착수하고 세번째 사람은 그다음 6개월후에 개발에 착수한다. 2명은 제품시험이 시작될 때 단계적으로 물러 나며 세번째 사람은 유지정비가 시작될 때 점차적으로 물러 난다.

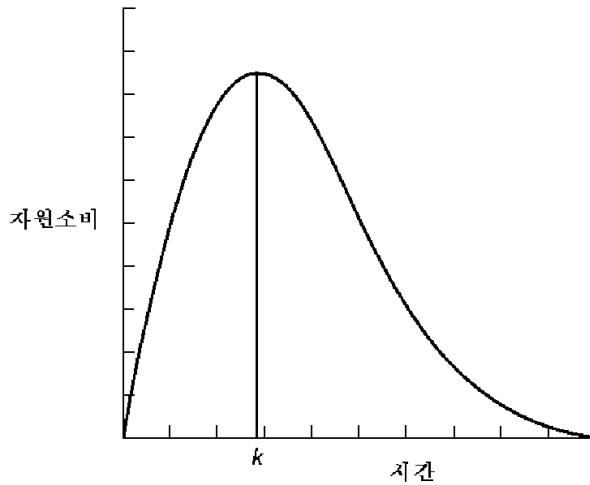



그림 9-7. 자원소비가 시간에 따라 변한다고 하는 레이레그곡선

자원수요가 시간에 의존한다는 사실은 개발성원들에게뿐아니라 콤퓨터시간, 지원쏘프트웨어, 콤퓨터하드웨어, 사무소설비능력에도 적용된다. 그리하여 쏘프트웨어프로젝트관리계획은 시간의 함수로 될것이다.

해야 할 과제는 두가지 부류로 갈라 볼수 있다. 첫째로, 프로젝트 전 기간에 걸쳐 계속되며 쏘프트웨어개발의 그 어떤 특정한 단계와 관계되지 않는 일이다. 그러한 작업을 프로젝트기능(*project function*)이라고 부른다. 실례로 프로젝트관리와 품질조종을 들수 있다. 두번째로, 제품개발에서 특정한 단계와 관련되는 작업이다. 즉 그러한 작업을 활동(*activity*) 또는 과제(*task*)라고 부른다. 활동은 정확한 시작과 끝나는 날자들이 있는 작업의 기본단위이다. 즉 콤퓨터시간 또는 일공수와 같은 자원들을 소비하고 예산, 설계문서, 일정계획, 원천코드 또는 사용자지도서와 같은 작업제품(*work product*)들로 된다. 결과 활동에는 과제 즉 관리책임에 따르는 작업의 가장 작은 단위인 과제의 모임이 포함된다. 쏘프트웨어프로젝트관리계획에는 세가지 종류의 작업들이 있다. 즉 프로젝트전기간에 걸쳐 수행되는 프로젝트기능, 활동(작업의 주요단위), 과제(작업의 부차적인 단위)들이 있다.

제품의 결정적인 측면은 작업제품들의 완성과 관련된다. 작업제품이 완성된다고 생각하는 날을 리정표(*milestone*)이라고 부른다. 작업제품이 실제로 리정표에 이르렀는가 하는것을 결정하기 위하여서는 그것에 대하여 우선 팀의 동료들과 관리자 또는 의뢰자에 의하여 수행되는 검토를 련속 진행하는것이다. 전형적인 리정표는 설계가 완성되고 검토에 통과되는 날이다. 일단 작업제품이 검토된후에 합의되면 그것은 기준선(*baseline*)으로 되며 5.8.2에서 설명한것처럼 공식적인 처리절차를 통하여 변경될수 있다.

사실상 단순히 제품 그자체보다도 작업제품에 더 많은것이 있다. 작업패키지는 바로 작업제품들뿐아니라 성원요구, 지속, 자원, 개별적인 책임의 이름, 작업제품을 위한 승인기준을 정의한다. 물론 비용은 계획에서 사활적인 구성요소이다. 구체화된 예산이 작성되여야 하며 시간의 함수로서 프로젝트기능들과 활동에 비용이 활당되여야 한다.

쏘프트웨어제품개발을 위한 계획작성과 관련한 문제는 다음절에서 서술한다.

9. 4. 쏘프트웨어프로젝트관리계획의 틀거리

그림 9-8은 IEEE규격1058.1 [IEEE 1058.1, 1987]에 규정된 쏘프트웨어관리계획(SPMP)의 부분제목들을 보여 주고 있다. 비록 SPMP를 작성하는 다른 많은 방법들이 있다고 하여도 IEEE규격에 따르는데는 명백한 우점이 있다.

1 소개
- 1.1 프로젝트 개괄
- 1.2 프로젝트 배포
- 1.3 쏘프트웨어프로젝트관리계획의 발전
- 1.4 참고자료
- 1.5 정의와 줄임말

2 프로젝트의 구성
- 2.1 공정모형
- 2.2 기업체의 구성
- 2.3 기업체의 경계와 호상결합
- 2.4 프로젝트책임성

3 관리공정
- 3.1 관리목적과 우선도
- 3.2 가정, 의존성, 제한
- 3.3 위험관리
- 3.4 감시 및 조종 기구
- 3.5 성원모집계획

4 기술공정
- 4.1 방법, 도구, 기술
- 4.2 쏘프트웨어문서작성
- 4.3 프로젝트지원기능

5 작업패키지, 일정계획, 예산
- 5.1 작업패키지
- 5.2 의존성
- 5.3 자원요구
- 5.4 예산과 자원할당
- 5.5 일정계획

보충적인 구성요소

그림 9-8. IEEE 쏘프트웨어프로젝트관리계획의 구성요소
[IEEE 1058.1, 1987]. [©1987 IEEE.]

첫째로, 그것이 쏘프트웨어개발에 참가하는 주요기업체들의 대표자들에 의해 작성된

표준이라는것이다. 공업과 대학들에서 자금을 투자하고 실무그룹과 심사팀성원들은 프로젝트관리계획작성에서 다년간의 경험을 가지고 있다. 표준은 이 경험을 통합한다. 두번째 우월성은 IEEE SPMP가 크기에 관계없이 모든 형태의 쏘프트웨어제품들을 리용하는데 적합하도록 설계된다는것이다. 그것은 특정한 개발공정모형을 강요하거나 특정한 방법을 규정하지 않는다. 본질상 계획은 특정한 응용분야를 위한 매개 기업체, 개발팀, 기법들에 의하여 적당하게 고쳐 진 내용들인 틀거리이다. 산업적인 토대에 기초한 이러한 틀거리를 견지함으로써 표준화의 우월성이 나타나게 된다. 결국 모든 쏘프트웨어개발성원들이 IEEE 쏘프트웨어프로젝트 관리계획형식에 익숙되게 될것이며 회사들은 새로운 개발성원들을 양성하는 비용을 절약하게 될것이다.

9. 5. IEEE 쏘프트웨어프로젝트관리계획

이제 계획틀거리자체를 자세히 설명하기로 한다. 본문에 있는 번호들과 표제들은 그림 9-8에 있는 서술항목들에 대응한다. 여기서 리용한 여러가지 용어들은 9.3에서 정의되였다.

1. 소개 SPMP에 대한 다섯개의 소절들은 프로젝트와 개발될 제품들에 대한 총체적인 견해를 주게 된다.

1.1. **프로젝트개괄** 여기서는 프로젝트목적, 배포하게 될 제품들, 활동 그리고 결과적인 작업제품들에 대한 간단한 설명을 준다. 이밖에 필요한 자원들과 기본일정계획 그리고 기본예산과 같은 리정표들이 목록으로 작성되게 된다.

1.2. **프로젝트의 배포** 의뢰자들에게 배포될 모든 항목들은 배포날자들과 함께 이 단계에서 목록작성되게 된다.

1.3. **쏘프트웨어프로젝트관리계획의 발전** 그 어떤 계획도 구체적으로 만들어 질수 없다. 임의의 다른 계획들과 마찬가지로 SPMP도 경험과 의뢰자측, 쏘프트웨어개발기업체내에서의 변화에 따라 계속 갱신되여야 한다. 이 절에서는 계획을 변화시키기 위한 형식적인 절차들과 기구들에 대하여 서술하였다.

1.4. **참고자료** SPMP에서 참고하는 모든 문서들이 참고자료안에 목록작성된다.

1.5. **정의들과 략어** 이 정보는 SPMP가 모든 사람들에게 똑같이 리해되도록 하게 한다.

2. 프로젝트의 구성 프로젝트개발기업체에 대한 부분은 4개로 되여 있는데 여기서는 쏘프트웨어개발공정과 개발자들의 기업체구조의 견지로부터 제품이 어떻게 개발되는가 하는데 대하여 자세히 서술하고 있다.

2.1. **공정모형** 공정모형은 제품설계 혹은 제품시험과 같은 활동들과 프로젝트관리 혹은 구성관리와 같은 프로젝트기능과 관련하여 자세히 서술된다. 여기서 기본적인 측면들은 리정표들, 기준선들, 검토, 작업제품, 배포에 관한 설명들이 포함된다.

2.2. **기업체의 구성** 여기서는 개발기업체의 관리구조가 서술된다. 기업체내의 권한과 책임한계를 명백히 하는것이 중요하다.

2.3. **기업체의 경계와 호상결합** 진공속에서는 그 어떤 프로젝트도 개발되지 못한다. 프로젝트개발성원들은 의뢰자측과 자기기업체의 다른 성원들과 호상작용하여야 한다. 더우

기 그 중간에 있는 청부업자들이 큰 프로젝트개발에 참가할수 있다. 프로젝트 그자체와 이러한 다른 구성요소들사이의 관리 및 경영상의 한계가 그어 져 있어야 한다. 그밖에 많은 쏘프트웨어개발기업체들은 두가지 류형의 그룹들 즉 단일한 프로젝트개발에 참가하는 그룹들과 기업체구성의 토대우에서 구성관리 및 쏘프트웨어품질보증과 같은 지원기능을 제공하는 지원그룹들로 갈라 진다. 프로젝트개발그룹과 지원그룹사이의 행정관리 및 경영상의 한계가 명백히 규정되여 있어야 한다.

2.4. **프로젝트책임성** SQA와 같은 매개 프로젝트기능과 제품시험과 같은 매개 활동을 책임질 개별적사람들을 확인하여야 한다.

3. 관리공정 이 부분에는 프로젝트가 어떻게 관리되는가 하는데 대하여 다섯가지로 설명한다.

3.1. **관리목적과 우선권** 관리를 위한 원리, 목표 그리고 우선권들을 서술한다. 계획에 대한 항목들의 형태들은 보고서작성의 빈도수와 기구, 요구중에서 상대적인 우선권, 프로젝트를 위한 일정계획과 예산 그리고 위험관리절차들이 포함된다.

3.2. **가정, 의존성, 제약조건** 명세서에 있는 가정들과 제약조건들이 여기에 포함된다.

3.3. **위험관리** 프로젝트와 관련한 여러가지 위험인자들이 이것들을 추적하는데 리용되는 방법들과 함께 여기서 목록작성된다.

3.4. **감시 및 조종기술** 검토 및 회계를 비롯하여 프로젝트를 위한 보고서작성기법들이 자세히 서술되여 있다.

3.5. **성원계획** 프로젝트개발에 참가하는 성원들은 중요한 자원을 이룬다. 필요한 성원들의 수와 류형들을 그들이 필요한 기간과 함께 렬거하여야 한다.

4. 기술공정 프로젝트의 기술적인 측면이 아래의 세가지로 설명된다.

4.1. **방법** 도구, 기법, 하드웨어와 쏘프트웨어의 기술적인 측면들이 여기에서 자세히 설명된다. 여기에는 제품이 실행될 목적체계들뿐아니라 제품을 개발하는데 리용될 콤퓨터체계들(하드웨어, 조작체계, 쏘프트웨어)이 포함된다. 필요한 다른 측면들로서는 리용될 개발기법, 시험방법, 팀의 구조, 프로그람작성언어 그리고 CASE도구들을 들수 있다. 그밖에 문서작성표준들과 코드작성표준들과 같은 기술표준들이 작업제품들을 개발하고 수정하기 위한 절차와 다른 문서들을 참고하여 포함되게 된다.

4.2. **쏘프트웨어문서작성** 여기에는 문서작성요구들 즉 리정표들과 기준선들 그리고 쏘프트웨어문서작성을 위한 검토들이 서술된다.

4.3. **프로젝트지원기능** 여기서는 시험계획을 비롯하여 구성배치관리와 품질보증과 같은 지원기능을 위한 계획들을 구체화한다.

5. 작업패키지, 일정계획, 예산 아래에서 작업패키지, 그것들사이의 호상의존성, 자원요구 그리고 련관된 예산할당들에 대하여 강조한다.

5.1. **작업패키지** 여기서는 작업패키지들을 활동들과 과제들로 분할된 련관된 작업제품들과 함께 자세히 서술한다.

5.2. **의존성** 모든 코드작성은 설계다음에 진행되며 통합 및 시험에 앞서서 진행한다. 일반적으로 작업패키지들에서의 호상의존성과 외부사건들에 대한 의존성이 존재한다. 여

기서 이 의존관계들을 서술한다.

5.3. **자원요구** 프로젝트를 완성하기 위하여서는 광범하고 다양한 자원들이 요구된다. 전체 자원들은 시간의 함수로 제공된다.

5.4. **예산과 자원할당** 여기서 렬거된 여러가지 자원들은 비용이 든다. 바로 자원이 시간이 지나감에 따라 변화되는것처럼 여러가지 구성요소자원들에 대한 예산할당도 변화된다. 여러가지 프로젝트기능들, 활동들, 그리고 과제들에 대한 자원들과 예산들의 할당이 진행된다.

5.5. **일정계획** 프로젝트의 매개 구성요소들을 위한 구체화된 일정계획이 주어 진다. 이러한 기본계획이 뒤따르게 된다. 즉 프로젝트가 제때에 그리고 예산범위내에서 완성될것을 요구한다.

추가적인 구성요소 일정한 프로젝트들을 위한 추가적인 구성요소들이 계획에 반영되여야 한다. IEEE틀거리에 비추어 그것들은 계획의 마감에 있게 된다. 추가적인 구성요소들에는 청부업자관리계획, 안전계획, 시험계획, 양성계획, 하드웨어조달계획, 설치계획 그리고 제품유지정비계획들이 포함된다.

9. 6. 시험계획작성

SPMP에서 자주 무시하게 되는 구성요소는 시험계획작성이다. 쏘프트웨어개발의 다른 모든 활동과 마찬가지로 시험이 계획되여야 한다. SPMP에는 시험을 위한 자원이 포함되여야 하며 구체화된 일정계획에서는 매 단계에서 수행하여야 할 시험이 명백히 지적되여야 한다.

시험계획이 없이는 프로젝트개발이 많은 면에서 잘못 진행될수 있다. 실례로 제품시험(2.6.1)기간에 SQA그룹은 명세서의 모든 측면들이 의뢰자에 의해서 계약명기되는것처럼 완성된 제품에서 실현되였는가를 시험하여야 한다. 이 과제수행에서 SQA그룹을 지원하는 좋은 방안은 개발이 추적가능하도록 요구하는것이다. 즉 명세서에 있는 매개 설명문을 설계의 부분들에 련결시키는것이 가능해야 하며 그리고 실례의 매개 부분을 코드에 명백히 반영하여야 한다. 이것을 달성하기 위한 한가지 기술은 명세서에 있는 매개 설명문들에 번호를 달아 주고 이 번호들이 설계와 결과코드에 반영되도록 하는것이다. 그러나 만일 시험계획이 이것을 진행하게 된다는것을 자세히 서술해 두지 않는다면 설계와 코드에 적당히 표시기호를 달아 놓을것까지는 필요 없다. 결과제품시험이 최종적으로 수행될 때에 SQA그룹이 제품이 명세서에 맞게 완전히 실현되였다는것을 확정하는것은 대단히 어려울것이다. 사실상 추적가능성은 요구사항확정단계와 함께 시작되여야 한다. 즉 요구사항확정문서에 있는 매개 설명문(혹은 신속원형의 매 부분)은 설계의 부분들에 련결되여야 한다.

검열의 한가지 위력한 측면은 검열기간에 발견된 오유들에 대한 구체적인 목록에 있다. 어느 한 팀이 제품의 명세서를 검열하고 있다고 하자. 6.2.3에서 설명한것처럼 오유들에 대한 목록은 두가지 방식으로 리용된다. 첫째로, 이 검열로부터 뽑아 낸 오유통계는 이전의 명세서검열에서 나온 축적된 평균고장통계와 비교하여야 한다. 이전의 기준으

로부터 벗어 난것들은 프로젝트안에서 문제가 있다는것을 의미한다. 둘째로, 현재 진행중의 명세서검열에서 나온 오유통계는 제품의 설계 및 코드검열들에 넘겨 져야 한다. 결국 만일 특정한 형태의 수많은 오유들이 있으면 명세서검열기간에 모두 검출되지 못할수 있으며 설계 및 코드검열들에서는 이 형태의 그 어떤 나머지오유들을 보충적으로 찾아낼수 있게 해준다. 그러나 만일 시험계획이 모든 오유들의 세부적인것들을 자세히 기록되여야 한다는것을 강조하지 않으면 이러한 과제는 수행될것 같지 않다.

코드모듈을 시험하는 한가지 중요한 방도는 코드가 명세서에 기초한 시험실레들로 시험을 진행하는 검은통시험(14.7)이다. SQA그룹의 성원들은 명세서를 쭉 읽고 코드가 명세서에 부합되는가를 검열하기 위하여 시험실레를 작성한다. 검은통시험실레들을 작성하는 가장 좋은 시간은 명세작성의 마감이다. 그러나 만일 시험계획이 검은통시험실레들을 이 시간에 선택하게 된다는것을 명백히 강조하지 않으면 아마 십중팔구는 다만 몇개의 검은통시험실레들만이 그후에 날림식으로 그러모아 만들어 지게 될것이다. 즉 SQA그룹이 그것의 모듈들이 전체로서 제품에 통합될수 있게 모듈들을 승인하도록 하기 위해 프로그람개발팀으로부터 요구가 높아 질 때에만 제한된 수의 시험실레들이 신속하게 수집된다. 결과 전체로서의 제품의 품질은 진통을 겪게 될것이다.

따라서 모든 시험계획은 무슨 시험을 진행하게 되고 또 그것이 언제 진행되여야 하며 또 그것이 어떻게 수행되게 되겠는가 하는것을 자세히 서술해 주어야 한다. 그러한 시험계획은 SPMP에 대한 4.3에서 본질적인 부분으로 되고 있다. 그것이 없이는 제품의 품질은 틀림없이 진통을 겪게 될것이다.

9. 7. 객체지향프로젝트의 계획작성

구조화파라다임을 리용한다고 하자. 개념적인 견지에서 볼 때 결과제품이 독립적인 모듈들로 구성되여 있다고 하더라도 그것은 일반적으로 하나의 큰 단위이다. 그와 상반되게 객체지향파라다임을 리용하는것은 수많은 상대적으로 독립인 보다 작은 구성요소들 즉 추상화의 분석 및 구성방식설계준위들에 있는 클라스들과 구체화된 설계 및 실현준위들에서의 객체들로 이루어 진 제품을 라렬화하는것으로 된다. 이것은 비용 및 기간타산들이 보다 더 작은 단위들을 위하여 더 쉽고 더 정확하게 계산될수 있다는 점에서 계획작성을 상당히 쉽게 해준다. 물론 제품에 대한 평가가 바로 그것의 부분들에 대한 평가의 합보다 더 많다는것을 고려하여야 한다. 개별적인 구성요소들은 완전히 독립이 아니다. 즉 그것들은 서로 호출할수 있다. 그리고 이러한 영향들은 무시하지 말아야 한다.

이 장에서 설명한 비용과 기간을 타산하기 위한 기술적인 방법들이 객체지향파라다임에 리용될수 있는가? COCCMO Ⅱ(9.2.4)은 객체지향을 비롯하여 현대적인 쏘프트웨어기술을 관리하기 위하여 설계되였다. 그러나 기능점들(9.2.1)과 중간 COCCMO(9.2.3)과 같은 초기의 척도는 어떤가? 중간 COCCMO의 경우에 일부 비용승수들에 대하여 보다 적은 변경을 진행하여야 한다[Pittman, 1993]. 그밖에 구조화파라다임의 평가도구들은 재리용되지 않으면 객체지향파라다임들에서 제대로 동작하지 않을것 같다. 재리용은 객체지향파라다임을 두가지 방식으로 입력한다. 개발기간에 현존구성요소들의 재

리용과 앞으로의 제품에 재리용하게 될 구성요소들의 계획적인 생산(현재의 프로젝트 개발기간에)을 입력하게 된다. 재리용의 두 형태들은 평가과정에 영향을 준다. 개발기간에 재리용은 명백히 비용과 기간을 줄인다. 이 재리용기능에 의한 절약을 보여 주는 공식들이 발표되였다[Schach, 1994]. 그러나 이 결과들은 구조화파라다임과 관련되여 있다. 현재 객체지향제품개발에 재리용이 전개될 때 비용과 기간이 어떻게 변하는가 하는것과 관련해서는 그 어떤 정보도 얻을수 없다.

이제 현재의 프로젝트의 부분들을 재리용하는 목적을 고찰하다.

재리용가능한 구성요소를 설계하고 실현하고 시험하여 문서작성하는것은 류사한 재리용불가능한 구성요소에 비하여 기간이 약 3배씩이나 오래 걸린다[Pittman, 1993]. 비용과 기간타산은 이 추가적인 노력을 통합하기 위하여 수정되여야 하며 전체로서의 SPMP는 재리용노력의 결과를 통합하기 위하여 조절되여야 한다. 따라서 두 재리용활동들은 반대방향으로 진행된다. 현존구성요소의 재리용은 객체지향제품을 개발하는데서 총체적인 노력을 줄인다. 반면에 앞으로의 제품들에서 재리용하기 위한 구성요소들을 설계하는것은 노력을 증대시킨다. 전망적으로 클라스들의 재리용으로 인한 절약이 원래의 개발비용보다 더 많을것으로 예상되며 일부 근거가 이미 확증되였다[Lim, 1994].

9. 8. 숙련에 대한 요구

숙련문제가 의뢰자와의 토의에서 제기되였을 때 공통적인 대답은 《우리는 제품이 완성될 때까지 숙련에 대해 걱정할 필요가 없다. 그다음 우리는 사용자들을 숙련시킬수 있다.》는것이다. 이것은 사용자들만이 숙련을 필요로 한다는것을 의미하는것으로서 어느 정도 유감스러운 대답이다. 사실상 숙련도 역시 쏘프트웨어계획작성 및 타산에서 숙련을 하게 되는것으로 하여 개발팀성원들에게도 필요할수 있다. 새로운 설계기법 혹은 시험절차들과 같은 새로운 쏘프트웨어개발기법들이 리용될 때 새로운 기법을 리용하는 모든 팀성원들을 숙련시켜야 한다.

객체지향파라다임도입은 주로 숙련의 귀결이다. 워크스테이션(*workstation*) 혹은 통합된 개발환경(15.8)과 같은 하드웨어 혹은 쏘프트웨어도구들의 도입에서도 숙련이 요구된다. 프로그람작성자들에게는 실현언어에서는 물론 제품개발을 위하여 리용하게 될 기계장치의 조작체계에 대한 숙련이 요구될수 있다. 문서작성숙련은 아주 많은 문서들에 대하여 품질이 낮다는것이 증명된것처럼 빈번히 무시된다. 콤퓨터조작공들에게는 틀림없이 새로운 제품을 다룰수 있는 숙련이 일정하게 요구된다. 즉 그들은 또한 새로운 하드웨어가 리용되면 추가적인 숙련을 요구할수 있다.

필요한 숙련은 여러가지 방식으로 받을수 있다. 가장 쉽고 좋은 숙련방식은 동업자들 혹은 고문들에 의해서 기업내부에서 수행하는 숙련이다. 많은 회사들은 다양한 숙련과정을 제공해 주며 대학들은 자주 야간숙련과정을 제공해 준다.

World Wide Web에 기초한 교육과정은 또 하나의 숙련과정으로 된다. 일단 숙련이 필요하다는것이 확정되면 숙련계획이 작성되고 그 계획은 SPMP안에 종합되여야 한다.

9. 9. 문서작성규격

쏘프트웨어제품개발은 광범하고 다양한 문서작성을 동반한다. 죠네스(Jones)는 크기가 약 50 KDSI되는 IBM내부의 상업용제품을 위하여 1,000개의 명령(KDSI)들마다 28페지의 문서들이 작성되며 똑 같은 크기의 상업용쏘프트웨어제품을 위해 KDSI당 약 68페지가 작성된다는것을 발견하였다. 조작체계 IMS/360 2.3판은 크기가 약 166KDSI이며 KDSI마다 157페지의 문서가 작성되였다. 문서는 계획작성, 조종, 재정 및 기술부서를 비롯하여 그 형식들이 다양하다[Jones, 1986a]. 이러한 형식의 문서들외에 원천코드자체는 문서의 형태이다. 즉 코드내에서의 주석들은 보충적인 문서를 이룬다.

쏘프트웨어개발노력의 상당한 몫이 문서작성에 돌려 진다. 63개의 개발프로젝트들과 25개의 유지정비프로젝트들에 대한 개관은 코드와 관련된 활동들에 소모된 모든 100h에 비해 볼 때 150h은 문서작성과 관련된 활동들에 소모되였다는것을 보여 주었다[Boehm, 1981]. 큰 TRW제품들에 한해서는 문서작성과 관련된 활동들에 바쳐 진 시간의 몫이 100h의 코드관련시간에 비해 200h까지 올라 갔다[Boehm et al., 1984].

모든 형태의 문서작성에는 규격이 필요하게 된다. 실례로 설계문서작성에서의 통일성은 팀성원들사이의 오해를 줄이고 SQA그룹성원들을 방조해 준다. 비록 문서작성규격들과 관련하여 새로운 성원들이 양성되여야 한다고 하더라도 기업체내에서 프로젝트와 프로젝트사이에 현존 성원들을 옮길 때에는 그 어떤 추가적인 숙련이 필요 없게 된다. 제품유지정비의 견지에서 볼 때 통일적인 코드작성규격들은 유지정비프로그람작성자들이 원천코드를 리해하도록 도와 준다. 규격은 이것들이 그들중 거의나 콤퓨터전문가가 아닌 광범한 개별적사람들이 읽고 리해해야 하기때문에 리용과 편람들을 위해서도 더 중요하다. IEEE는 사용자편람을 위한 규격(쏘프트웨어사용자문서작성을 위한 IEEE규격 1063)을 발표하였다.

계획작성공정의 한 부분으로서 쏘프트웨어개발기간에 작성된 모든 문서들을 위한 규격들이 만들어 져야 한다. 이 규격들은 SPMP에 통합된다. 쏘프트웨어시험문서를 위한 IEEE규격[ANSI/IEEE 829, 1991]과 같은 현존규격들이 리용되는데서는 SPMP(참고자료)의 1.4에 규격이 목록작성되여 있다. 만일 개발노력을 위해 어떤 규격이 특별히 만들어 지면 그것은 4.1에서 보여 주었다. 문서작성은 쏘프트웨어개발노력의 본질적인 측면이다. 문서작성이 없이는 제품이 유지정비될수 없기때문에 진정한 의미에서 제품은 곧 문서이다. 문서작성노력을 자세히 계획작성하고 그다음 그 계획이 준수되도록 하는 것은 쏘프트웨어개발을 성과적으로 진행할수 있게 하는 중요한 요인으로 된다.

9. 10. 계획작성 및 타산을 위한 CASE도구

중간 COCOMO와 COCOMO Ⅱ를 자동화하는 수많은 도구들을 리용할수 있다. 파라메터값이 수정되는 경우 계산속도를 위해 중간 COCOMO의 여러개의 실현들이 Lotos1-2-3 혹은 Excel과 같은 표처리프로그람작성언어들로 개발되였다. 계획 그자체를 개발하고 갱신하기 위하여서는 문서처리프로그람이 필요하다.

관리정보도구들도 역시 계획작성에 쓸모가 있다. 실례로 큰 쏘프트웨어기업체가 150명의 프로그람작성자들을 가지고 있다고 하자. 일정계획작성도구는 계획작성자가 어느 프로그람작성자에게 특정한 과제를 할당해 주고 어느것이 현재의 프로젝트를 위해 리용될수 있는가를 기록하는것을 도와 줄수 있다.

보다 일반적인 형식의 관리정보들도 역시 요구들과 수많은 상업적으로 리용할수 있는 관리도구들이 계획작성 및 타산과정을 지원하고 전체적으로 개발공정을 감시조종하도록 하는데 리용할수 있다. 이러한것들가운데는 MacProject와 Microsoft Project가 있다.

9.11. 쏘프트웨어프로젝트관리계획의 시험

이 장의 앞부분에서 지적한것처럼 쏘프트웨어프로젝트관리체계에서의 결함은 개발자들에 있어서 심중한 재정적의미를 가질수 있다는것이다. 개발기업체가 프로젝트의 비용 혹은 기간을 과대평가하거나 과소평가하지 않는것이 매우 중요하다. 이것으로 하여 전체 SPMP는 의뢰자에게 평가결과를 주기전에 SQA그룹에 의해서 시험되여야 한다. 계획을 시험하는 가장 좋은 방법은 11.11에서 설명한 명세서검열과 류사하게 계획검열을 진행하는것이다.

계획검열팀은 비용과 기간에 특별한 주의를 돌리면서 SPMP를 구체적으로 검토하여야 한다. 리용된 척도에 관계없이 위험성을 좀더 줄이기 위해 계획작성팀성원들이 자기들의 평가를 확인하자마자 비용과 기간타산들이 SQA그룹성원들에 의해 독자적으로 평가되여야 한다.

요 약

이 장의 기본주제는 쏘프트웨어개발공정(9.1)에서의 계획작성의 중요성이다. 쏘프트웨어프로젝트관리계획의 중요한 구성요소는 기간과 비용(9.2)을 타산하는것이다. 기능점(9.2.1)들을 비롯해서 제품의 크기를 타산하기 위한 여러개의 척도가 제기되였다. 다음으로 비용타산을 위한 여러가지 척도를 설명하였다. 특히 중간 COCOMO(9.2.3)와 COCOMO Ⅱ(9.2.4)를 설명한다. 쏘프트웨어프로젝트관리계획의 세가지 구성요소들 즉 진행할 작업, 그것을 진행하는데 리용될 자원들 그리고 그것에 적용한 비용을 9.3에서 설명하였다.

한가지 특별한 SPMP인 IEEE규격을 9.4에서 륜곽적으로 설명하고 9.5에서 구체적으로 서술하였다. 그다음 계획작성시험(9.6)과 관련한 절들이 있고 계속하여 객체지향프로젝트에 대하여서는 계획작성(9.7)과 요구사항확정의 숙련과 문서작성규격들, 계획작성공정(9.8과 9.9)을 위한 절들을 취급하였다. 9.10에는 계획작성 및 타산을 위한 CASE도구들이 서술되여 있다. 이 장은 끝으로 쏘프트웨어프로젝트관리(9.11)에 대한 시험과 관련한 자료들을 서술하였다.

보 충

웨인비그(Weinberg)의 4권으로 된 저서 [Weinberg, 1992, 1993, 1994, 1997]는 [Whitten, 1995, and Thayer, 1997]과 같이 쏘프트웨어관리의 많은 측면들에 대하여 상세한 정보를 제공해 주고 있다. 쏘프트웨어관리에 대한 새로운 견해들은 문헌 [Reifer, 2000]에 있다. 쏘프트웨어프로젝트를 위한 척도는 문헌 [Weller, 1994]에서 론의되였다.

객체지향파라다임의 관리를 위하여서는 문헌 [Pittman, 1993, and Nesi, 1998]을 참고할수 있다. 개발자의 생산성에 대한 객체지향틀거리[8.5.2]의 결과는 문헌 [Moser and Nierstrasz, 1996]에서 론의하였다. *IEEE Software*의 1996년 7월호에는 대규모프로젝트들의 관리에 대한 수많은 기사들이 들어 있다. 이러한것들로서는 문헌 [Charette, 1996]이 있는데 이것은 대규모프로젝트들을 관리하는데 관계되는 위험들을 론의하였다. 위험관리에 대한 또 다른 기사는 문헌 [Gemmer, 1997]이다. *IEEE Computer*의 1996년 9월호에는 객체지향프로젝트들의 관리에 대한 기사들이 들어 있다. 특별한 관심사로 되는것은 문헌 [Sparks, Benner, and Faris, 1996, and Williams, 1996]이다.

쏘프트웨어프로젝트관리계획을 위한 IEEE규격1058.1에 관한 보충적인 정보를 얻자면 규격 그자체를 자세히 보아야 한다[IEEE 1058.1, 1987]. [Sackman, Erikson, amd Grant, 1968]에는 새크맨(Sackman)의 저서가 서술되여 있다. 보다더 구체적인 문헌은 [Sackman, 1970]이다. 문헌 [Shepperd and Ince, 1994]를 비롯하여 쏘프트웨어과학에 대한 많은 비판적인 개관들이 발표되였다. 기능점들에 대한 정보는 문헌 [Low and Jeffrey, 1990]에서 찾아 볼수 있다. 기능점에 대한 개선된 세밀한 분석은 문헌 [Symons, 1991]에서 고찰하였다. 기능점들에 대한 비평들이 문헌 [Kitchenham, 1997]에서 고찰하였다. 다른 분석들은 문헌 [Jeffrey, Low, and Barnes, 1993, and Kitchenham and Känsälä, 1993]에 있다. 비용타산공정을 개선하기 위한 제의는 문헌 [Lederer and Prasad, 1992]에서 론의하였다. 기능점들의 우점과 약점들은 문헌 [Furey and Kitchenham, 1997]에서 지적하였다. 기능점의 모든 측면에 대한 정보는 문헌 [Boehm, 1997]에서 종합적으로 주었다.

중간 COCOMO를 실현하기 위한 구체적인 세부들과 함께 그것을 위한 리론적인 조정은 문헌 [Boehm, 1981]에 있다. 보다 짧은 판본은 문헌 [Boehm, 1984b]에서 설명하고 있다. COCOMO II는 [Boehm et al., 2000]에서 고찰하였다.

문헌 [Myers, 1989]는 아주 큰 쏘프트웨어프로젝트들을 위한 개발시간을 타산하기 위한 척도에 관한 정보를 제공해 주고 있다. 다양한 업무자료처리제품들을 위한 쏘프트웨어생산성자료가 문헌 [Maxwell and Forselius, 2000]에 제시되여 있다. 리용된 생산성의 단위는 시간당 기능점수들이다. 주어 진 제품의 문서페지수들을 평가하기 위한 공식은 문헌 [Jones, 1994b]에서 서술하였다.

문 제

9.1. 일부 쏘프트웨어기업체들이 왜 리정표(*milestone*)를 무거운 짐(*millstone*)이라고 말한다고 생각하는가?

9.2. 당신은 네더버그쏘프트웨어개발회사에 있는 쏘프트웨어공학자이다. 한해전에 당신의 경영자는 당신의 다음제품이 9개의 파일들, 49개의 흐름, 87개의 공정들을 포함할것이라고 발표하였다.

(1) FFP 척도를 리용하여 그 크기를 확정하시오.
(2) 네더버그쏘프트웨어개발회사를 위해 식 (9.2)에 있는 실수 d는 932딸라 된다고 확정되였다. FFP 척도는 비용타산을 얼마로 예측했는가?
(3) 최근에 제품이 134,500딸라의 비용으로 완성되였다. 이것은 개발팀의 생산성에 대해 무엇을 말해 줄수 있는가?

9.3. 목적하는 제품에는 7개의 단순한 입력, 11개의 보통의 입력 그리고 8개의 복잡한 입력들이 있다. 40개의 보통출력들, 5개의 단순한 질문들, 18개의 평균주파일들 그리고 12개의 복잡한 대면부들이 있다. 조절되지 않는 기능점들(*UFP*)을 결정하라.

9.4. 만일 문제 9.3의 제품을 위한 총체적인 영향의 정도가 49이라면 기능점들의 수를 결정하라.

9.5. 당신은 그의 약점에도 불구하고 왜 코드행들(Loc혹은 KDSI)이 제품크기의 척도로 그렇게 널리 리용된다고 생각하는가?

9.6. 당신은 자료기지의 크기가 대단히 높다고 타산하고 쏘프트웨어도구들의 리용이 낮은것으로 타산된것을 제외하고 명목상인 79-KDSI가 내장된 제품을 개발하는것을 책임졌다. 중간 COCOMO를 리용하면 월공수로 평가된 노력은 얼마인가?

9.7. 당신은 두개의 45-KDSI 단순한 방식의 제품을 개발할 책임을 졌다. 제품 P1가 특별히 높은 복잡성을 가지고 있고 제품 P2이 특별히 낮은 복잡성을 가지고 있는것외에 모든 측면에서 그것들은 둘 다 명목상이다. 제품을 개발하기 위하여 당신은 바라던대로 2개의 팀을 가지게 된다. 팀 A는 대단히 높은 분석능력과 응용경험 그리고 프로그람작성능력을 가지고 있다. 팀 A는 또한 높은 가상기계숙련과 프로그람작성언어경험을 가지고 있다. 팀 B는 5개의 모든 속성들에 한에서 대단히 낮은 수준으로 평가된다.

(ㄱ) 만일 팀 A가 제품 P_1를 개발하고 팀 B가 제품 P_2을 개발하다고 하면 총체적인 노력은 월공수로 얼마인가?
(ㄴ) 만일 팀 B가 제품 P_1를 개발하고 팀 A가 제품 P_2을 개발한다면 전체 노력은 월공수로 얼마인가?

(ㄷ) 두개의 선행한 성원배치들중에서 어느 배치가 더 의미 있는가? 당신의 직감이 COCOMO의 예측의 지지를 받고 있는가?

9.8. 당신은 모든 측면에서 명목상인 54-KDSI 단순한 방식의 제품을 개발할 책임을 졌다.

(1) 평균월공수당 9,400딸라의 비용을 가정하면 타산된 프로젝트의 비용은 얼마인가?

(2) 당신이 전체 개발팀을 프로젝트의 시발점에서 단념하였다. 당신은 운이 좋아 명목상의 팀을 대단히 경험이 많고 능력이 있는 팀으로 교체할수 있었다. 그러나 평균 월공수당 비용이 12,200딸라로 오를것이다.
당신은 인원변동의 결과로 얼마나 많은 돈을 벌게 된다고(혹은 손해보리라고) 기대하는가?

9.9. 당신은 큰 화물차운송회사를 위해 가장 비용이 효률적인 로정을 계산하기 위한 쏘프트웨어제품을 새롭게 연구된 알고리듬을 리용하여 개발할 책임을 지고 있다. 중간 COCOMO을 리용하여 당신은 제품의 비용이 430,000딸라일것이라는것을 확정한다. 그러나 시험하여 당신은 팀의 한 성원에게 기능점들을 리용한 노력을 평가해 보라고 요구한다. 그는 기능점척도가 당신의 COCOMO예측보다 수배씩이나 더 큰 890,000딸라의 비용을 예측하였다고 보고한다. 이제 당신은 무엇을 하는가?

9.10. 레이레그분산(식 (9.9))이 $t = k$일 때 최대값을 얻는다는것을 보여 준다. 해당한 자원소비를 찾으시오.

9.11. 제품유지정비계획이 IEEE SPMP의 《추가적인 구성요소》이라고 생각한다. 모든 중요한 제품들이 유지정비되고 유지정비비용이 평균 제품을 개발하는 비용의 두배나 된다는것을 생각하면 이것을 어떻게 조절하겠는가?

9.12. (과정안상 목표) 부록 1에 있는 브로드랜즈지역 아동병원프로젝트를 생각해 보자. 순수 부록 1에 있는 정보에 기초하여 비용과 기간을 타산하는것이 왜 가능하지 못한가?

9.13. (쏘프트웨어공학독본) 교원은 문헌 [Nesi, 1998]의 복사본을 나누어 줄것이다. 당신은 객체지향쏘프트웨어제품들을 관리하는 가장 도전적인 측면들이 무엇이라고 생각하는가?

참 고 문 헌

[Albrecht, 1979] A. J. ALBRECHT, "Measuring Application Development Productivity," *Proceedings of the IBM SHARE/GUIDE Applications Development Symposium,* Monterey, CA, October 1979, pp. 83–92.

[ANSI/IEEE 829, 1991] "Software Test Documentation," ANSI/IEEE 829-1991, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1991.

[Boehm, 1981] B. W. BOEHM, *Software Engineering Economics,* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Boehm, 1984b] B. W. BOEHM, "Software Engineering Economics," *IEEE Transactions on Software Engineering* **SE-10** (January 1984), pp. 4–21.

[Boehm, 1997] R. BOEHM (EDITOR), "Function Point FAQ," ourworld.compuserve.com/homepages/softcomp/fpfaq.htm, June 25, 1997.

[Boehm et al., 1984] B. W. BOEHM, M. H. PENEDO, E. D. STUCKLE, R. D. WILLIAMS, AND A. B. PYSTER, "A Software Development Environment for Improving Productivity," *IEEE Computer* **17** (June 1984), pp. 30–44.

[Boehm et al., 2000] B. W. BOEHM, C. ABTS, A. W. BROWN, S. CHULANI, B. K. CLARK, E. HOROWITZ, R. MADACHY, D. REIFER, AND B. STEECE, *Software Cost Estimation with COCOMO II,* Prentice Hall, Upper Saddle River, NJ, 2000.

[Charette, 1996] R. N. CHARETTE, "Large-Scale Project Management *Is* Risk Management," *IEEE Software* **13** (July 1996), pp. 110–17.

[Devenny, 1976] T. DEVENNY, "An Exploratory Study of Software Cost Estimating at the Electronic Systems Division," Thesis No. GSM/SM/765–4, Air Force Institute of Technology, Dayton, OH, 1976.

[Furey and Kitchenham, 1997] S. FUREY AND B. KITCHENHAM, "Function Points," *IEEE Software* **14** (March/April 1997), pp. 28–32.

[Gemmer, 1997] A. GEMMER, "Risk Management: Moving beyond Process," *IEEE Computer* **30** (May 1997), pp. 33–43.

[Halstead, 1977] M. H. HALSTEAD, *Elements of Software Science,* Elsevier North-Holland, New York, 1977.

[IEEE 1058.1, 1987] "Standard for Software Project Management Plans," IEEE 1058.1, Institute of Electrical and Electronic Engineers, New York, 1987.

[Jeffrey, Low, and Barnes, 1993] D. R. JEFFREY, G. C. LOW, AND M. BARNES, "A Comparison of Function Point Counting Techniques," *IEEE Transactions on Software Engineering* **19** (May 1993), pp. 529–32.

[Jones, 1986a] C. JONES, *Programming Productivity,* McGraw-Hill, New York, 1986.

[Jones, 1987] C. JONES, Letter to the Editor, *IEEE Computer* **20** (December 1987), p. 4.

[Jones, 1994b] C. JONES, "Cutting the High Cost of Software 'Paperwork'," *IEEE Computer* **27** (October 1994), pp. 79–80.

[Kemerer, 1993] C. F. KEMERER, "Reliability of Function Points Measurement: A Field Experiment," *Communications of the ACM* **36** (February 1993), pp. 85–97.

[Kemerer and Porter, 1992] C. F. KEMERER AND B. S. PORTER, "Improving the Reliability of Function Point Measurement: An Empirical Study," *IEEE Transactions on Software Engineering* **18** (November 1992), pp. 1011–24.

[Kitchenham, 1997] B. KITCHENHAM, "The Problem with Function Points," *IEEE Software* **14** (March/April 1997), pp. 29, 31.

[Kitchenham and Känsälä, 1993] B. KITCHENHAM AND K. KÄNSÄLÄ, "Inter-Item Correlations among Function Points," *Proceedings of the IEEE 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 477–80.

[Lederer and Prasad, 1992] A. L. LEDERER AND J. PRASAD, "Nine Management Guidelines for

Better Cost Estimating," *Communications of the ACM* **35** (February 1992), pp. 51–59.
[Lim, 1994] W. C. LIM, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software* **11** (September 1994), pp. 23–30.
[Low and Jeffrey, 1990] G. C. LOW AND D. R. JEFFREY, "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering* **16** (January 1990), pp. 64–71.
[Maxwell and Forselius, 2000] K. D. MAXWELL AND P. FORSELIUS, "Benchmarking Software Development Productivity," *IEEE Software* **17** (January/February 2000), pp. 80–88.
[Moser and Nierstrasz, 1996] S. MOSER AND O. NIERSTRASZ, "The Effect of Object-Oriented Frameworks on Developer Productivity," *IEEE Computer* **29** (September 1996), pp. 45–51.
[Myers, 1989] W. MYERS, "Allow Plenty of Time for Large-Scale Software," *IEEE Software* **6** (July 1989), pp. 92–99.
[Nesi, 1998] P. NESI, "Managing OO Projects Better," *IEEE Software* **15** (July/August 1998), pp. 50–60.
[Norden, 1958] P. V. NORDEN, "Curve Fitting for a Model of Applied Research and Development Scheduling," *IBM Journal of Research and Development* **2** (July 1958), pp. 232–48.
[Pittman, 1993] M. PITTMAN, "Lessons Learned in Managing Object-Oriented Development," *IEEE Software* **10** (January 1993), pp. 43–53.
[Putnam, 1978] L. H. PUTNAM, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering* **SE-4** (July 1978), pp. 345–61.
[Reifer, 2000] D. J. REIFER, "Software Management: The Good, the Bad, and the Ugly," *IEEE Software* **17** (March/April 2000), pp. 73–75.
[Sackman, 1970] H. SACKMAN, *Man–Computer Problem Solving: Experimental Evaluation of Time-Sharing and Batch Processing,* Auerbach, Princeton, NJ, 1970.
[Sackman, Erikson, and Grant, 1968] H. SACKMAN, W. J. ERIKSON, AND E. E. GRANT, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM* **11** (January 1968), pp. 3–11.
[Schach, 1994] S. R. SCHACH, "The Economic Impact of Software Reuse on Maintenance," *Journal of Software Maintenance: Research and Practice* **6** (July/August 1994), pp. 185–96.
[Shepperd, 1988a] M. J. SHEPPERD, "An Evaluation of Software Product Metrics," *Information and Software Technology* **30** (No. 3, 1988), pp. 177–88.
[Shepperd and Ince, 1994] M. SHEPPERD AND D. C. INCE, "A Critique of Three Metrics," *Journal of Systems and Software* **26** (September 1994), pp. 197–210.
[Sparks, Benner, and Faris, 1996] S. SPARKS, K. BENNER, AND C. FARIS, "Managing Object-Oriented Framework Reuse," *IEEE Computer* **29** (September 1996), pp. 52–61.
[Symons, 1991] C. R. SYMONS, *Software Sizing and Estimating: Mk II FPA,* John Wiley and Sons, Chichester, U.K., 1991.
[Thayer, 1997] R. H. THAYER, *Software Engineering Project Management,* 2nd ed., IEEE Computer Society Press, Los Alamitos, CA, 1997.
[van der Poel and Schach, 1983] K. G. VAN DER POEL AND S. R. SCHACH, "A Software Metric for Cost Estimation and Efficiency Measurement in Data Processing System Development," *Journal of Systems and Software* **3** (September 1983), pp. 187–91.
[Weinberg, 1992] G. M. WEINBERG, *Quality Software Management: Systems Thinking,* Volume 1, Dorset House, New York, 1992.
[Weinberg, 1993] G. M. WEINBERG, *Quality Software Management: First-Order*

Measurement, Volume 2, Dorset House, New York, 1993.
[Weinberg, 1994] G. M. WEINBERG, *Quality Software Management: Congruent Action,* Volume 3, Dorset House, New York, 1994.
[Weinberg, 1997] G. M. WEINBERG, *Quality Software Management: Anticipating Change,* Volume 4, Dorset House, New York, 1997.
[Weller, 1994] E. F. WELLER, "Using Metrics to Manage Software Projects," *IEEE Computer* **27** (September 1994), pp. 27–34.
[Weyuker, 1988b] E. WEYUKER, "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering* **14** (September 1988), pp. 1357–65.
[Whitten, 1995] N. M. WHITTEN, *Managing Software Development Projects,* and ed., John Wiley and Sons, New York, 1995.
[Williams, 1996] J. D. WILLIAMS, "Managing Iteration in OO Projects," *IEEE Computer* **29** (September 1996), pp. 39–43.

제 2 편
쏘프트웨어생명주기의 단계

제2편에서는 쏘프트웨어생명주기의 단계들에 대하여 깊이 있게 서술한다. 매 단계들에 대하여 그 단계의 설명요구뿐만아니라 그에 고유한 CASE도구들, 척도 및 시험기법들이 제시된다.

제10장에서는 요구사항확정단계에 대하여 고찰한다. 이 단계의 목적은 의뢰자의 실제의 요구를 확정하는것이다. 이 목적을 달성하기 위한 한가지 기법은 신속원형작성법이다. 이 장에서 각이한 류형의 신속원형작성법과 기타 요구분석기법들이 고찰된다.

일단 요구가 결정되면 그 다음단계는 명세서를 작성하는것이다. 여기에 대하여서는 제11장《명세작성단계》에서 고찰한다. 명세작성에 관한 기본방법들인 비형식적, 준형식적 및 형식적명세작성방법들이 제시되며 매 방법들에 대한 실례들도 서술된다. 구조화체계분석, 유한상태기계, 페트리망을 포함한 기법들이 깊이 있게 서술되고 실례연구에 의하여 례증되며 여러가지 기법들에 대한 Z. A비교를 제시한다.

제11장에서 서술하는 모든 명세작성기법들은 구조화파라다임에 근원을 두고 있다. 객체지향명세작성법은 제12장 《객체지향분석단계》에서 서술한다. 이 객체지향명세작성기법은 앞장에서 고찰한 구조화명세작성기법의 변종으로서 제시된다.

제13장 《설계단계》에서는 객체지향설계뿐만아니라 자료호출분석, 트랜잭션업무분석과 같은 고전적방법들을 포함하여 여러가지 설계방법들을 비교한다. 이 장에서는 특히 실례연구를 포함하여 객체지향설계에 주목을 돌리며 비교와 대조에 중점을 둔다.

실현과 관련한 론의는 제14장과 제15장에서 진행된다. 제14장은 실현단계를 고찰하고 있는데 여기에는 프로그람작성언어의 선택, 4세대프로그람언어, 좋은 프로그람작성관계, 프로그람작성규격들이 포함된다. 실현단계와 통합화단계는 병렬로 수행되여야 하는데 이것은 제15장 《실현 및 통합단계》의 주제로 된다.

제2편은 제16장 《유지정비단계》로 결속된다. 제16장의 기본화제는 유지정비의 중요성과 설명요구이다. 유지정비에 관한 관리문제가 약간 자세하게 고찰된다. 제2편을 결속하면서 쏘프트웨어개발공정의 모든 단계들과 그 매개 단계와 관련된 요구들, 이 요구를 만족시키기 위한 방도들에 대하여 명백한 리해를 가지게 될것이다.

제１０장. 요구사항확정단계

쏘프트웨어개발팀성원들이 어떤 쏘프트웨어제품의 사명에 대하여 동의하지 않는한에서는 그 제품이 주어 진 운영비범위내에서 제기간에 개발될 가능성이 거의 없다. 이와 같은 합의를 이룩하기 위한 첫단계는 의뢰자의 현재상황을 될수록 정확히 분석하는것이다. 실례로 《의뢰자들은 자기들이 가지고 있는 수동설계체계가 락후하기때문에 콤퓨터지원설계체계를 요구한다.》고 말한다면 그것은 적당치 않다. 그것은 개발팀이 현재의 수동체계가 가지고 있는 결함을 정확히 알지 못하는 한에서는 새로운 콤퓨터화된 체계의 상황 역시 《락후》해 질 가능성이 크기때문이다. 이와 류사하게 만일 어떤 개인용콤퓨터제조업자가 새로운 조작체계를 개발하려고 하는 경우 그 첫단계는 회사의 현존조작체계를 평가하여 그것이 왜 불만족스러운가를 주의 깊게 정확히 분석하는것이다. 극단적인 실례를 든다면 문제가 판매업자의 심중에만 존재하는가, 누가 그 조작체계의 불매에 대하여 비난하는가, 그 조작체계의 사용자들이 그 체계의 가능성과 신뢰성에 대한 환상에서 깨여 났는가를 아는것은 극히 사활적인 문제로 된다. 현재의 상황을 명백히 리해하여야만 개발팀은 새 제품이 무엇을 할수 있는가 하는 중대한 문제에 답변할수 있다. 이 문제에 대한 답변과정은 요구사항확정단계에서 수행되게 된다.

일반적으로 생기는 오해는 요구사항확정단계에서 개발자들이 의뢰자가 무슨 쏘프트웨어를 원하는가를 결정하여야 한다고 하는것이다. 이와 달리 요구사항확정단계의 실지 목표는 의뢰자가 무슨 쏘프트웨어를 필요로 하는가를 결정하는것이다. 문제는 바로 많은 의뢰자들이 자기들이 무엇을 필요로 하는가를 모르고 있다는데 있다. 더우기 대부분의 의뢰자들은 개발팀성원들보다 콤퓨터에 조예가 깊지 못하기때문에 무엇이 필요되는가에 대한 훌륭한 착상을 가지고 있는 의뢰자조차 자기의 착상을 개발자에게 정확히 전달하기 어려울수도 있다.

1967년에 미국의 대통령후보자 죠지 롬니(George Romney)는 자기의 발언에서 한번에 너무 많은 실수를 하였다. 그는 어느 한 기자회견을 소집하고 다음과 같은 설명을 발표하였던것이다.

《내 생각에는 당신이 내가 말한것을 자기가 리해하였다고 믿고 있는것 같은데 내 생각에는 당신이 자기가 들은것이 내가 말하려고 하였던것이 아니라는것을 깨닫지 못한것 같다.》

이와 같은 변명은 요구분석에서도 그대로 적용된다. 즉 개발자들은 의뢰자들의 요구를 듣지만 그들이 듣는것은 의뢰자가 말하는것이 아니다.

제3장에서는 이와 같은 정보전달에 의한 문제를 해결하기 위한 한가지 방도가 신속원형작성을 진행하는것이라는것을 지적하였다. 이 장에서는 요구사항확정단계를 더욱 자세히 서술하며 각이한 요구분석기법들의 장점과 약점들에 대하여 론의한다.

먼저 요구도출문제부터 고찰하기로 한다.

10. 1. 요구도출

의뢰자의 요구를 발견하는 과정을 요구도출(또는 요구포착)이라고 부른다. 일단 초기 요구모임이 결정되면 그것을 세련시키고 확장하는데 이 과정을 요구분석이라고 부른다. 요구사항확정단계는 일반적으로 한명 또는 그이상의 요구사항확정팀성원들이 목적하는 제품에서 무엇이 필요한가를 결정하기 위하여 한명 또는 그이상의 의뢰자들과 면담하는 것으로부터 시작된다.

의뢰자의 요구를 도출하기 위하여서는 요구사항확정팀성원들이 응용령역 즉 제안된 쏘프트웨어제품이 리용되게 될 일반령역에 정통하여야 한다. 례를 들어 은행경영이나 병간호에 대하여 먼저 일정한 정도로 정통함이 없이 은행일군이나 간호원에게 의미 있는 문제를 제기한다는것은 쉽지 않다. 그러므로 요구분석팀의 매 성원들이 선차적으로 해야 할 일은 자기가 해당 일반령역에 대한 경험이 없는 한에서는 그 응용령역에 정통하는것이다. 목적하는 쏘프트웨어에 대한 의뢰자와 잠정적인 사용자들과 정보교환을 진행할 때 정확한 용어를 사용하는 문제가 특별히 중요하다. 결국 면담자들이 그 령역에 고유한 전문술어를 리용하지 않는다면 어떤 전문령역에 종사하는 사람들과 진지하게 작업하기가 어려워 진다. 보다 중요하게는 부적당한 용어들의 리용은 오해를 산생시켜 결국은 오작제품이 배포되는데로 이어 질수도 있다. 실례로 비전문가에게는 버팀대, 들보, 대들보, 지주목과 같은 단어들이 동의어처럼 생각되지만 토목기사에게는 이 단어들이 서로 차이나는 용어들이다. 만일 개발자가 토목기사가 이 4개의 용어를 정확히 구분하여 리용하고 있다는것을 식별하지 못한다면 그리고 만일 토목기사가 개발자는 이 용어들사이의 차이를 알고 있다고 가정하고 있다면 개발자는 이 4개용어들을 같은것으로 취급할수도 있다. 결국 콤퓨터지원교량설계프로그람은 다리붕괴를 초래할수 있는 오유를 포함하게 된다. 콤퓨터전문가들은 매 프로그람에 기초해서 결심이 체택되기전에 그 프로그람의 결과들이 자세히 조사되기를 바란다. 그러나 콤퓨터에 대한 대중적인 신뢰가 증대되고 있는 현실은 이와 같은 검사가 진행될 가능성에 의거하는것이 명백히 어리석은 일이라는것을 말해주고 있다. 그러므로 전문술어에서의 오해가 쏘프트웨어개발자들의 무책임성을 고소하는데로 이어 질수 있다고 생각하는것은 의미가 없다.

전문술어와 관련된 이러한 문제를 해결하기 위한 한가지 방도는 용어해설집을 만드는것이다. 요구사항확정팀이 응용령역에 대하여 학습하는 과정에 초보적인 용어들이 용어해설집에 들어 간다. 그다음 용어해설집은 요구사항확정팀성원들이 새로운 술어를 만날 때마다 갱신된다. 이와 같은 용어해설집은 의뢰자와 개발자사이의 혼돈을 줄일뿐만아니라 개발팀성원들사이의 오해를 줄이는데도 유익하다.

일단 요구사항확정팀이 령역에 정통하게 되면 그 다음단계는 의뢰자의 요구에 대한 결정을 시작하는것 즉 요구도출을 진행하는것이다. 1차적인 도출기법은 면담을 진행하는 것이다.

10. 1. 1. 면담

요구사항확정팀성원들은 자기들이 의뢰자와 미래의 제품사용자들로부터 련관된 정보

를 모두 도출하였다고 확신할 때까지 의뢰자팀성원들과 면담을 진행해야 한다. 면담에는 두가지 기본형식 즉 구조화된 면담과 비구조화된 면담이 있다. 구조화된 면담에서는 전문적이면서 미리 계획되고 고정된 문제들이 제출된다. 실례로 의뢰자는 회사가 몇명의 판매원을 고용하고 있는가, 얼마나 빠른 대답시간을 요구하는가 하는 문제를 제기할수 있다. 비구조화된 면담에서는 면담대방이 말을 하도록 하면서 고정되지 않은 문제들이 제출된다. 실례로 의뢰자에게 《왜 현재의 제품이 만족스럽지 못합니까?》라고 묻는다면 의뢰자는 업무와 관련한 많은 상황을 설명할수도 있다. 이러한 일부 사실들은 만일 면담이 구조화된 경우에는 드러나지 않을수도 있는것들이다.

이와 동시에 면담을 지나치게 비구조화하는것은 좋은 방법이 아니다. 의뢰자에게 《현재의 제품에 대하여 말하시오.》라고 말한다면 적당치 못한 정보들을 많이 만들어내게 될것이다. 그러므로 질문은 면담대방이 면담자가 요구하는 정보범위내에서 폭 넓은 답변을 줄수 있도록 유도하는 방식으로 제시되여야 한다.

면담을 솜씨 있게 이끌어 나가는것은 언제나 쉬운것은 아니다. 첫째로, 면담자는 응용령역에 정통하여야 한다. 둘째로, 면담자가 이미 의뢰자의 요구를 파악하였다면 의뢰자팀의 어떤 성원과 면담할 때 아무런 리득도 얻지 못한다. 면담자는 비록 다른 수단에 의하여 이미 듣었거나 알았다고 하여도 의뢰회사나 의뢰자들의 요구 그리고 구성하려는 제품의 잠정적리용과 관련하여 이미 알고 있는 개념들을 될수록 나타내지 않으면서 면담대방이 말해야 할것을 주의깊게 듣는 방향애서 매 면담에 대하여야 한다.

면담이 계속되면 면담자는 면담결과를 제시하는 서면보고서를 준비하여야 한다. 면담대방에게 보고서의 부본을 반드시 주어야 한다. 왜냐하면 면담대방이 문장내용을 명백히 하거나 주요 항목들은 훑어 보려고 할수도 있기때문이다.

10. 1. 2. 대본작성

대본작성은 요구분석을 진행하는 또 한가지 기법이다. 대본작성은 사용자가 일정한 목적을 달성하기 위하여 목적하는 제품을 활용할수 있게 하는 방법이다. 실례로 목적하는 제품이 몸까기계획프로그람이라고 하자. 하나의 가능한 대본은 영양학자가 환자의 나이, 성별, 무게, 키 기타 개인자료를 입력할 때 무엇이 발생하는가를 서술하는것이다. 프로그람은 그다음 그 환자에 대한 표본차림표를 찍어 낸다. 이 대본이 미래의 프로그람사용자에게 제시될 때 영양학자는 그 차림표가 당뇨병환자, 채식주의자, 유당을 싫어 하는 사람들과 같은 특수한 음식요구를 가지고 있는 환자들에게 적당치 않을 수도 있다는것을 재빨리 지적한다. 개발자는 임의의 식사차림표를 찍어 내기에 앞서 사용자가 특수한 음식을 요구할수 있도록 이 대본을 수정한다. 대본의 리용은 사용자로 하여금 자기들의 수요를 요구분석자들에게 전달할수 있도록 한다.

몸까기계획작성프로그람의 실례에서 나아가서 이 프로그람이 영양학자가 환자의 키를 센치메터로 입력할 때 그것을 인치로 내보낼것을 요구한다고 가정하자. 이것은 하나의 례외실례이다. 대본은 기대하는 사건렬들뿐만아니라 모든 례외적사건들도 포함하여야 한다. 대본은 각이한 방식으로 묘사될수 있다. 한가지 기법은 대본을 구성하고 있는 작용들을 단순하게 렬거하는것이다. 이것은 11장에서 설명한다. 또 한가지 기법은 문서화상적재

판 즉 사건렬을 묘사하는 선도를 설정하는것이다. 문서화상적재판은 문서원형[Retting, 1994] 즉 련관된 화면과 사용자의 답변을 묘사하고 있는 종이판렬로 생각할수 있다.

그러나 무슨 방법이 선택되든지간에 대본은 출발상태, 기대하는 사건렬, 최종상태, 기대하는 사건에 례외적인 사건들을 모두 묘사하여야 한다.

대본은 다른 여러가지 방법들에서도 쓸모 있다.

1. 대본은 제품의 거동을 사용자에게 알기 쉬운 방식으로 보여 줄수 있다. 이것은 몸까기계획프로그람에서처럼 추가적인 요구를 드러내게 할수 있다.
2. 대본은 사용자들이 리해할수 있기때문에 대본의 활용은 의뢰자와 사용자들이 요구분석과정의 전반에 걸쳐 능동적인 역할을 놀수 있게 한다. 결국 요구분석단계의 목표는 의뢰자의 실제적요구를 도출하는것이며 이러한 정보의 유일한 원천은 의뢰자와 사용자들이다.
3. 대본(또는 보다 정확하게는 리용실례)은 객체지향분석에서 중요한 역할을 논다. 이에 대하여서는 12.3에서 자세히 서술한다.

이제 요구도출에 관한 기타 기법들을 고찰하기로 하자.

10.1.3. 기타 요구도출기법

요구를 도출하기 위한 다른 방법은 질문서를 련관된 의뢰기구성원들에게 보내는것이다. 이 기법은 수많은 개별적사람들의 견해를 결정할 필요가 있을 때 유용하다. 더우기 주의 깊게 생각해 낸 서면답변은 면담자가 제기한 질문에 구두로 즉시에 답변하는것보다 더 정확할수 있다. 그러나 면담자가 초기답변을 주의 깊게 듣고 그것을 확장한 질문을 제기하는 방법으로 진행해 나가는 비구조화된 면담은 일반적으로 주의 깊게 작성된 질문서보다 훨씬 더 좋은 정보를 준다. 질문서는 사전에 계획되기때문에 질문이 답변에 따라 제안될 방도는 없다.

특히 기업환경에서 요구를 도출하는 또 다른 방법은 의뢰자가 리용하는 각이한 형식을 조사하는것이다. 실례로 어떤 출판사에서 리용하고 있는 출판형식은 출판량, 종이로라 크기, 습도, 잉크온도, 종이의 장력 등을 반영할수도 있다. 각이한 분야들에서 이 형식을 리용하여 인쇄작업의 흐름과 련관된 출판공정에서의 중요한 단계들에 개선을 가져 올수 있다. 조작절차와 일반서술과 같은 기타 문서들도 무엇을 어떻게 수행해야 하는가를 찾아 내기 위한 강력한 수단으로 될수 있다. 의뢰자가 현재 어떻게 경영해 나가고 있는가를 고려하고 있는 이와 같은 종합적인 정보는 의뢰자의 요구를 결정하는데서 특별히 도움이 될수 있다. 그렇기때문에 의뢰자의 문서작성은 주의 깊게 음미해 보는 과정을 절대로 홀시하지 말아야 한다. 왜냐하면 그것이 의뢰자의 요구를 정확히 판단할수 있게 하는 하나의 정보자원으로 되기때문이다. 이와 같은 정보를 획득하기 위한 보다 새로운 방법은 작업장안에 비데오카메라를 설치하고 그 곳에서 진행되고 있는 과정을 정확하게 기록하는것이다. 이 방법을 적용하는데서 나서는 한가지 어려운 문제는 테프를 분석하는데 오랜 시간이 걸린다는것이다. 일반적으로 한명 또는 그이상의 요구분석성원들은 카메라가 한시간동안 기록한 테프를 재생하는데 한시간을 소비하여야만 한다. 이 시간에 관측

된것을 평가하는데 필요한 시간이 더 합쳐 진다. 보다 심중하게는 종업원들이 카메라를 자기들의 사생활에 대한 부당한 침해로 볼수 있기때문에 이 기법은 나쁜 결과를 초래하는것으로 알려 져 있다.

중요한것은 요구분석팀이 모든 종업원들과 충분히 협동하는것이다. 만일 사람들이 협박을 받는다든가, 피로움을 겪는다고 느끼게 되면 필요한 정보를 획득하는것은 매우 어렵다. 카메라를 도입하기전에 있을수 있는 위험을 주의 깊게 고찰하여야 하며 또는 그러한 문제가 생긴 경우에는 성난 종업원들을 진정시킬수 있는 다른 임의의 방법을 취하여야 한다. 일단 초기요구모임이 얻어 지면 그 다음단계는 그것을 실현시키는것인데 이 과정을 요구분석이라고 부른다.

10. 2. 요구분석

이 단계에서 요구사항확정팀은 하나의 예비요구모임을 가지게 된다. 이러한 요구들에는 두가지 류형 즉 기능적요구와 비기능적요구들이 포함된다. 기능적요구들은 목적하는 쏘프트웨어의 기능성과 관련되여 있다. 실례로 《매 배우들의 공연료금은 1998년 5월에 제정한 CMS공식을 리용하여 공연목록자료로부터 계산될것이다.》를 들수 있다. 비기능적명세서는 믿음성과 유지정비성과 같은 목적한 쏘프트웨어의 성질을 규정하거나 쏘프트웨어가 실행될 환경과 관련되여 있다. 실례로 《모든 막대코드(*bar code*)들은 Mach/Zor나 ASRCA입력장치를 리용하여 읽혀 질것이다.》를 들수 있다.

쏘프트웨어가 추적가능해야 한다는것은 본질적인 문제이다. 즉 명세서, 설계, 코드를 포함하여 요구문서안의 매 문장들은 추적할수 있어야 한다. 이런 방식으로 SQA그룹은 요구문서안의 매 문장이 실현되였는가 그것이 정확히 수행되였는가를 검사할수 있다. 추적가능성을 실현하기 위하여서는 요구문서안의 매 문장에 번호를 매길 필요가 있다.

예비요구문서안의 모든 항목들은 우선권을 가지도록 의뢰자에게 제공된다. 의뢰자(또는 의뢰자팀)는 본질적인것, 몹시 필요한것, 필요한것 등등의 범주를 리용하여 매개의 예비요구들을 분석한다. 이 과정을 진행하는 기간에 어떤 요구들은 부정확하거나 상관이 없다는것이 명백해 질수 있는데 이러한 요구들을 교정하거나 삭제한다. 다음단계는 예비요구문서를 더욱 세련시키는것이다.

우선 요구사항확정팀성원들은 무엇을 생략하겠는가를 검정하기 위하여 각이한 개별적면담대방들과 요구문목록을 토의한다. 그다음 가장 정확하고 강력한 요구분석기법은 신속원형작성방법이기때문에 하나의 신속원형을 작성한다. 이에 대하여 다음절에서 고찰한다.

10. 3. 신속원형작성방법

신속원형작성은 쏘프트웨어를 신속히 작성하는것인데 그것은 목적하는 제품의 중요한 가능성들을 보여 준다. 실례로 종합주택의 관리를 지원하는 쏘프트웨어는 사용자가 새 거주자에 대한 세부사항을 입력할수 있게 하는 입력화면과 매달 거주보고서를 인쇄하는 기능을 병합하여야 한다. 이러한 상황들이 신속원형작성에 병합된다. 그러나 오유검

사능력, 파일갱신부분프로그람, 주택사용료금계산들은 포함되지 않을수도 있다. 관건적인 문제는 하나의 신속원형이 입력화면이나 보고서와 같은 의뢰자가 볼수 있는 기능성들을 반영하지만 파일갱신과 같은 《은폐》된 상황들은 생략하고 있다는것이다(신속원형작성법을 고찰하는 다른 방법에 대하여서는 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

어떤 제품의 중요한 측면을 보여 줄 목적으로 모형을 구성할데 대한 착상은 오래전에 나왔다. 실례로 도메니코 크레스티(Domenico Cresti; 그는 이딸리아의 치안티구역 파씨그나노읍에서 출생하였기때문에 《II파씨그나노》로 알려 져 있다.)가 1618년에 그린 그림은 미겔란젤로가 로마법왕 파울로 4세에게 증정한 로마의 페터스거리에 대한 나무로 만든 설계모형을 보여 주고 있다. 이러한 건축모형은 규모가 방대하였다. 즉 건축가 브라만테(Bramante)가 작성한 앞선 설계계획은 한면의 길이가 20피트이상에 달하였다.

건축모형은 각이한 목적에 많이 리용되였다. 우선 크레스티가 그린 그림에서와 같이 모형은 어떤 프로젝트에 자금을 투자하는데 의뢰자의 관심을 끌도록 하는데 리용되였다. 이것은 신속원형을 의뢰자의 실지 수요를 결정하는데 리용하는것과 류사하다. 다음으로 건축설계가 진행되기 이전시대에 모형은 건축가들에게 건물의 구조를 보여 주고 석공들에게는 건물을 어떻게 장식해야 하는가를 지적하는데 리용되였다. 이것은 10.4에서 설명한바와 같이 사용자대면부에 대한 신속원형을 작성하는 방법과 류사하다.

그러나 건축모형과 쏘프트웨어의 신속원형을 지나치게 밀접히 비교하는것은 그리 좋지는 않다. 신속원형은 의뢰자의 요구를 도출하기 위하여 요구사항확정단계에서 리용된다. 건축모형과는 달리 신속원형은 건축설계나 세부구조를 제시하는데는 리용되지 않는다. 설계는 두단계 지나서 즉 설계단계에서 만들어 진다.

제품의 의뢰자와 예정된 사용자들은 신속원형으로 실험을 진행하며 한편 개발팀성원들은 요점을 찾아 쥔다. 자기들의 경험에 기초하여 사용자들은 개발자들에게 신속원형이 자기들의 요구를 얼마나 만족시키는가를 말하며 보다 중요하게는 개선할 필요가 있는 부분들을 찾아 낸다. 개발자들은 의뢰자의 요구들이 신속원형에 정확하게 교갑된다는것을 량측이 확증할 때까지 신속원형을 변화시킨다. 그다음 신속원형은 명세서를 작성하기 위한 기초로 리용된다.

신속원형작성모형의 중요한 측면은 《조기》라는 말에 구현되여 있다. 궁극적으로는 원형을 가능한대로 빨리 작성하는것이다. 결국 신속원형작성의 목적은 의뢰자가 제품에 대하여 더빨리 더잘 리해하도록 하는것이다. 신속원형이 힘들게 동작하고 그것이 매번 몇분간 폭주되거나 화면설계가 완전하지 못하다 하여도 문제로 되지 않는다. 신속원형작성의 목적은 의뢰자와 개발자가 그 제품이 해야 할바에 대하여 될수록 빨리 합의할수 있도록 하는것이다. 그렇기때문에 신속원형에서의 임의의 불완전한 측면들은 그것들이 신속원형의 기능성을 심히 손상시키지 않으며 또한 쏘프트웨어제품의 동작에 대하여 그릇된 표상을 주지 않는다는 조건하에서 무시될수 있다.

신속원형작성모형의 두번째로 중요한 측면은 신속원형을 변화시킬수 있도록 작성하

여야 한다는것이다. 만일 신속원형의 첫번째 판본이 의뢰자가 요구하는것이 아니라면 그 원형은 의뢰자의 요구를 더잘 만족시키는 두번째 판본으로 신속히 전환되여야 한다. 신속원형작성 전 과정에 걸쳐서 개발을 신속하게 진행하기 위하여 Smalltalk, Prolog, Lisp와 같은 4세대프로그람언어(4GLs)들과 해석형언어들을 리용한다. 현재 광범히 리용되고 있는 신속원형작성언어들로는 visual C++와 J++뿐만아니라 HTML들과 Perl도 들수 있다. 일부 해석형언어들에 주의가 돌려 졌지만 고전적인 원형작성이라는 관점에서 보면 이것은 적당하지 못하다. 문제점은 다음과 같다. 어떤 주어 진 언어가 신속원형작성에 리용될수 있는가? 신속원형은 재빨리 변화될수 있는가? 만일 이 두가지 질문에 대한 답변이 긍정이라면 그 언어는 신속원형작성을 위한 후보로서 적당하다고 할수 있다.

이제 신속원형을 객체지향모형과 결합하여 사용하는 방향으로 전환하여 보면 IBM회사가 진행한 세가지 서로 다른 객체지향프로젝트들은 구조화된 모형을 리용하는 프로젝트들에 비하여 뚜렷한 개선을 보여 주었다[Capper, Colgate, Hunter, and James, 1994].

이와 같은 프로젝트들로부터 얻게 되는 한가지 문제는 객체지향생명주기에서 될수록 빨리 신속원형을 작성하는것이 중요하다는것이다.

신속원형은 어떤 쏘프트웨어제품에 대한 사용자대면부를 개발할 때에도 매우 효과적이다. 이러한 리용방법과 관련해서는 다음 절에서 고찰한다.

10. 4. 인간적인자

쏘프트웨어제품의 의뢰자와 미래의 사용자가 모두 사용자대면부에 대한 신속원형과 호상작용한다. 사용자들이 인간 콤퓨터대면부(HCI)로 실험하도록 하는것은 최종제품이 변경될 위험성을 크게 감소시킨다. 특히 이러한 실험은 모든 쏘프트웨어제품들의 사활적 목적인 사용자친절성을 달성할수 있게 한다.

사용자친절성이라는 술어는 인간이 쏘프트웨어제품과 쉽게 정보교환을 할수 있다는것을 의미한다. 만일 사용자가 쏘프트웨어제품을 사용하는 방법을 배우기 어렵거나 화면이 혼돈되거나 번거럽다고 느끼게 된다면 사용자는 더는 그 제품을 리용하지 않거나 잘못 리용하게 될것이다. 이러한 문제를 해소하기 위하여 차림표구동프로그람이 도입되였다. 즉 《계산을 수행하시오.》든가 《봉사료금에 관한 보고서를 인쇄하시오.》와 같은 명령을 입력하는것 대신에 사용자는 다음과 같은 가능한 응답모임들가운데서 단순한 선택만 진행하면 된다.

1. 계산을 수행하시오.
2. 봉사료금에 관한 보고서를 인쇄하시오.
3. 그라프보기를 선택하시오.

이 실례에서 사용자는 1, 2, 3을 입력하여 대응하는 명령들을 호출한다.

현재는 단순한 문장렬을 현시하지 않고 HCI는 도형방식을 리용한다. 윈도우, 아이콘, 내리펼침차림표(*pull-doun menu*)들은 도형사용자대면부(GUI)의 요소들이다. 윈도우즈체계의 파잉으로 인하여 X window와 같은 표준체계가 개발되였다. 또한 《지적과 찰칵(*point and click*)》과 같은 선택이 표준방식으로 되고 있다. 사용자들은 마우스를 움직여 화면우

의 카소르를 희망하는 응답("*point*")에로 움직이며 마우스단추를 눌러("*click*") 그 응답을 선택한다.

그러나 목적하는 프로그람제품이 현대적인 기술을 리용할 때에도 설계자들은 그 제품을 인간이 리용하게 된다는것을 잊어서는 안된다. 달리 말하면 HCI설계자들은 문자의 크기, 대문자사용, 색, 행길이, 화면의 행수와 같은 인간적인자들을 고려하여야 한다.

앞에서 설명한 차림표는 인간적인자가 적용되는 또 한가지 실례이다. 만일 사용자가 《3. 그라프보기를 선택하시오.》를 선택하면 다른 선택목록을 제시하는 차림표가 나타난다. 차림표구동체계를 주의 깊게 설계하지 않으면 상대적으로 단순한 조작을 실현하기 위하여 사용자가 긴 차림표렬을 다루어야 할 위험성이 생기게 된다. 이와 같은 지연현상은 사용자를 노엽히며 때로는 그것이 부적당한 차림표를 선택하게 할수도 있다. 또한 HCI는 사용자가 제일 웃준위차림표에로 되돌아 가지 않고 앞서 진행한 선택을 변화시켜 다시 시작할수 있도록 설계하여야 한다. 이러한 문제는 GUI가 리용될 때에도 존재할수 있다. 왜냐하면 많은 도형사용자대면부들은 본질상 매력적인 화면형식으로 현시되는 차림표렬이기때문이다.

때때로 단일한 사용자대면부가 모든 사용자들을 만족시킬수도 있다. 실례로 만일 어떤 쏘프트웨어제품을 콤퓨터전문가와 콤퓨터에 대한 경험이 없는 중학교중퇴생이 리용하기로 되여 있다면 두개의 서로 다른 HCI모임을 설계하는것이 좋다. 이 두가지 대면부는 예정된 사용자들의 숙련수준과 심리학적측면에 맞추어 만들어 진다. 이와 같은 기술은 여러가지 수준의 정교성을 요구하는 사용자대면부들을 병합함으로써 확장될수 있다. 만일 제품이 사용자가 자주 실수하거나 방조수단들을 련속적으로 쏘프트웨어호출하는것으로 인하여 보다 정교성이 낮은 사용자대면부에 만족스러워 할것이라는 론리에 기초하여 만들어 진다면 사용자에게는 응당 자기의 숙련수준에 보다 적합한 화면이 보여 지게 된다. 그러나 사용자가 제품에 보다 정통하게 되면 보다 적은 정보를 제공하여 주는 간소화된 화면이 현시되게 되며 보다 빨리 목적을 달성할수 있게 한다. 이러한 자동화된 방법은 사용자의 실패를 줄이고 생산성을 증가시킨다[Schach and Wood, 1986].

HCI를 설계하는 과정에 인간적인자들을 고려하면 학습시간의 감소와 오유률의 저하를 비롯한 많은 리익을 얻을수 있다. 비록 방조(*help*)수단들은 언제나 제공되여야 하지만 그것들은 정교하게 설계된 HCI보다 덜 활용된다. 이것 역시 제품의 생산성을 증가시킨다. 어떤 제품 또는 제품그룹전환에서 HCI현시의 단일성을 보장하는것은 사용자로 하여금 이전에 본적이 없는 화면들을 리용하는 방법을 직관적으로 알수 있게 한다. 왜냐하면 그러한 화면들은 그들이 이미 정통하고 있는 화면과 류사하기때문이다. 마킨토쉬쏘프트웨어설계자들은 이와 같은 원리를 고려하여 제품을 만들고 있다. 이것이 바로 마킨토시용프로그람이 그처럼 사용자에게 친숙하게 하는 원인들중의 하나이다. 사용자에게 친절한 HCI를 설계하는것이 필요하다는것은 하나의 단순한 상식으로 되였다. 이것이 사실인가 아닌가에 관계없이 매 제품에 대한 HCI신속원형이 작성된다는것이 본질적이다. 예정된 프로그람사용자들은 HCI신속원형을 실험하고 설계자에게 목적하는 제품이 실지로 사용자에게 친절한가 아닌가 즉 설계자들이 필수적인 인간적인자들을 고려하였는가를 알려 줄수 있다.

다음 두개의 절에서 표면상 매력이 있지만 위험을 동반하고 있는 기타 신속원형작성

모형이 론의된다.

10. 5. 명세작성기법으로서의 신속원형작성

3.3에서 서술되고 그림 3.3에서 보여 준바와 같이 신속원형작성모형의 전통적인 형식을 그림 10-1에서 보여 주었다(실현 및 통합단계는 일반적으로 병렬로 수행된다.). 신속원형은 의뢰자의 요구가 정확하게 도출되였다는것을 결정하기 위한 수단으로서 리용된다.

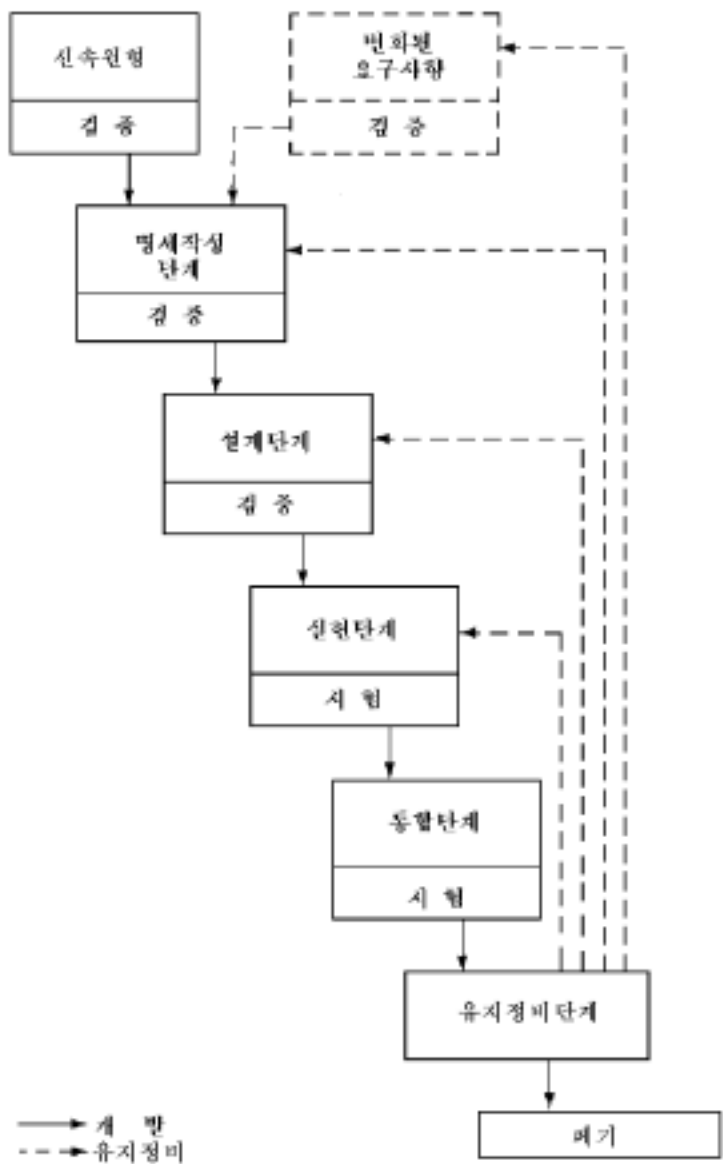



그림 10-1. 신속원형작성모형

일단 의뢰자가 명세서에 수표하면 신속원형작성모형은 필요 없게 된다(그러나 학습한 경험은 보류되여 다음 개발공정에서 리용된다.). 두번째 방법은 신속원형 그자체를 명세서나 명세서의 중요부분으로 리용하여 명세서를 필요 없게 하는것이다. 이와 같은 두번째 류형의 신속원형작성모형은 그림 10-2에서 보여 주었다. 이 방법은 속도와 정확성을 모두 보장하여 준다. 이 방법에서는 서면명세서를 작성하는 시간이 필요 없게 되고 애매성, 탈락, 모순과 같은 명세작성과 관련된 난점들이 발생하지 않는다. 그대신에 신속원형이 명세서를 구성하기때문에 여기서 해야 할 일은 제품이 신속원형이 하는 작업을 진행할것이라는것을 서술하고 파일갱신, 암호화, 오유조종과 같은 제품이 지원해야 할 모든 추가적속성들을 렬거하는것이다.

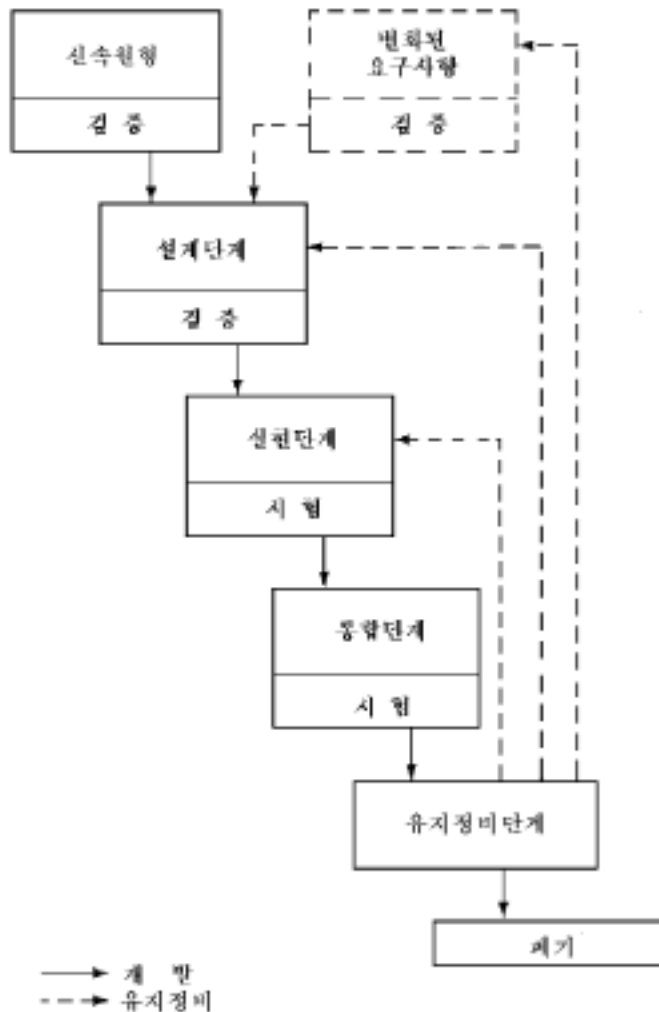



그림 10-2. 신속원형을 설계명세서로 제공하는 신속원형작성과정

이러한 형식의 신속원형작성모형은 중요한 약점을 가질수 있다. 만일 개발자의 원만

한 책임수행여부에 관한 문제에서 의견이 불일치한다면 신속원형을 개발자와 의뢰자사이의 계약서의 합법적인 설명으로 취할수는 없을것 같다. 이러한 리유로 하여 신속원형을 유일한 명세서로서 리용하지 말아야 한다. 프로그람이 내부적으로 개발되는 경우(즉 의뢰자와 개발자가 같은 기업체의 성원인 경우)에도 사정은 마찬가지이다. 비록 은행의 투자관리부의 책임자가 자료처리부를 법에 고소할것 같지는 않다고 하여도 의뢰자와 개발자사이의 의견불일치는 기관내에서도 쉽게 발생할수 있다. 그렇기때문에 이러한 현상을 막기 위하여 쏘프트웨어개발자들은 신속원형을 쏘프트웨어를 내부적으로 개발하는 경우라 하여도 명세서로서 리용하지 말아야 한다.

신속원형을 서면명세서로서 대치하지 말아야 할 두번째 리유는 유지정비와 관련한 잠재적인 문제때문이다. 제16장에서 서술된바와 같이 유지정비는 모든 문서작성이 유효하고 갱신되는 경우에조차 헐치 않은 일이다. 만일 아무런 명세서도 없다면 유지정비를 진행하는것은 전혀 불가능하게 된다. 이 문제는 요구를 변경시켜야 하는 보다 강한 조건에서 특별히 심각하게 된다. 서면명세서가 없으면 유지정비팀에게 현재의 명세서에 대한 명확한 진술이 주어 지지 않기때문에 새로운 명세서를 반영하는 설계문서들을 변경시키는 일이 매우 어려워 진다.

이와 같은 두가지 원인으로 하여 신속원형은 단순히 요구분석기법 즉 의뢰자의 실제 요구가 정확히 도출되였다는것은 확인하기 위한 한가지 수단으로서 리용하여야 한다. 그 다음 서면명세서들은 신속원형을 토대로 하여 만들어야 한다.

10. 6. 신속원형의 재리용

앞에서 고찰한 두가지 신속원형작성모형에서 신속원형은 요즈음에는 쏘프트웨어개발과정에서 리용되지 않는다. 그리 좋지는 못하지만 한가지 다른 처리방법은 신속원형을 제품으로 완성될 때까지 발전시키고 세련시키는것이다. 이 과정은 그림 10-3에 보여 주었다. 리론적으로 이 방법은 쏘프트웨어를 빨리 개발하는데로 이끌어 간다. 결국 신속원형을 구성하고 있는 코드들을 버리는것 대신에 거기에 지식을 추가시켜 나가면서 신속원형을 최종적인 제품으로 전환시킨다. 그러나 실천적으로 이 과정은 그림 3-1에서 보여 준 구성 및 수정(*built-and-fix*)방법과 매우 류사하다. 구성 및 수정모형에서와 마찬가지로 이러한 형식의 신속원형작성모형에서 제기되는 첫번째문제는 신속원형을 세련시키는 과정에서 작업하고 있는 제품들에 대하여 변경을 가해야 한다는것이다. 이것은 그림 1-5에서 보여 준바와 같이 품이 많이 드는 처리방법이다. 두번째문제는 신속원형을 작성할 때 1차적인 목적은 작성속도라는것이다. 신속원형은 주의 깊게 전문화되고 설계되며 실현되는것이 아니라 조급하게 결합되는것이다. 명세서와 설계문서가 없으면 결과적인 코드는 유지정비하기가 어렵고 품이 많이 든다. 하나의 신속원형을 작성한 다음 그것을 버리고 제품을 처음부터 다시 설계하는것은 비경제적인것 같지만 그것은 단기적관점에서나 장기적관점에서 모두 신속원형을 품질 좋은 쏘프트웨어제품으로 전환시키려고 하는것보다는 훨씬 값이 눅다[Brooks, 1975].

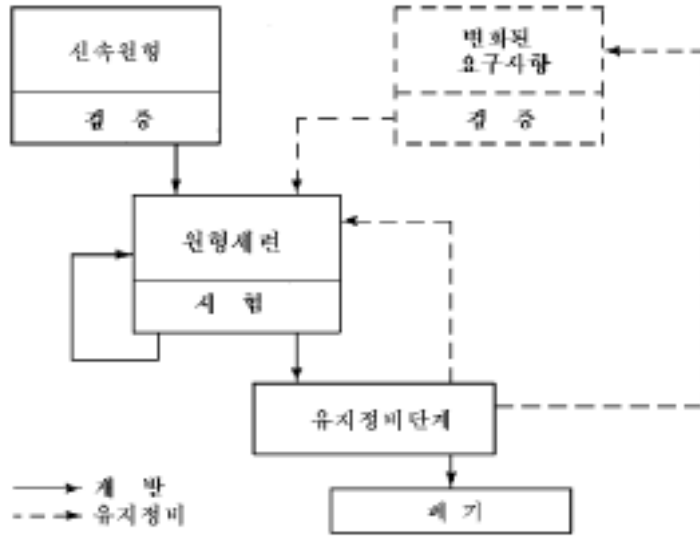



그림 10-3. 신속원형작성모형의 그릇된 판본

신속원형을 버려야 할 또 하나의 리유는 제품의 성능 특히는 실시간체계에 관한 문제때문이다. 시간제약이 만족된다는것을 담보하기 위하여서는 제품을 신중하게 설계하여야 한다. 이와는 대비적으로 신속원형은 의뢰자에게 중요한 기능성을 현시하도록 구성된다. 즉 이때 성능에 대하여서는 론의하지 않는다. 결국 만일 신속원형을 정식제품으로 세련시키려고 하면 응답시간 및 기타 시간적제약이 만족되지 않을수 있다.

신속원형이 버려 지고 제품이 정확하게 설계되고 실현되였다는것을 확인하기 위한 한가지 방법은 신속원형을 제품과는 다른 언어로 작성하는것이다. 실례로 의뢰자는 제품이 Java언어로 작성되여야 한다고 규정할수도 있다. 만일 신속원형을 HTML로 실현된다면 그것은 버려야 한다. 먼저 신속원형이 HTML로 실현하고 목적하는 프로그람이 해야 할 모든 사항 또는 대부분의 사항들에 의뢰자가 만족할 때까지 그것을 세련시킨다. 다음으로 신속원형을 구성할 때 획득한 지식과 숙련에 기초하여 제품을 설계한다. 마지막으로 설계를 Java언어로 실현하고 시험을 마친 제품을 의뢰자에게 일반적인 방식으로 넘겨 준다.

그러나 신속원형 또는 특수하게 신속원형의 부분들을 세련시키는것이 허용되는 한가지 실례가 있다. 즉 신속원형의 부분들이 콤퓨터에 의하여 생성된다면 이러한 부분들은 최종제품으로 리용될수 있다. 실례로 사용자대면부는 신속원형의 중요한 응용국면으로 된다(10.3).

화면생성프로그람, 보고서생성프로그람(5.4)과 같은 CASE도구들이 사용자대면부를 생성하는데 활용될 때 이와 같은 신속원형의 부분들은 실지로 생산성과 품질이 좋은 쏘프트웨어의 부분으로서 리용될수 있다.

신속원형을 《랑비》하지 않으려는 욕망은 결국 일부 단계들에서 리용하고 있는 변화된 류형의 신속원형작성모형이 생겨 나게 했다. 이 모형에서 관리자측은 신속원형이 만들어 지기전에 그 부분들이 다른 쏘프트웨어요소들과 같은 품질보증검사를 거친다는 조건하에서 최종제품에 리용될수 있다는것을 결정한다. 그러므로 신속원형이 완성된후에

개발자들이 계속 리용하려고 하는 부분들은 설계 및 코드시험을 거쳐야 한다. 이 방법은 신속원형작성법을 초월한다. 실례로 설계 및 코드시험을 거친 충분히 질이 좋은 요소들은 신속원형에서는 찾아 볼수 없다. 더우기 설계문서들은 고전적인 신속원형의 부분으로 되지 않는다. 그러나 이와 같은 혼성방법은 신속원형작성에 투자한 시간과 자금을 회복하려고 하는 일부 개발기업체들의 주목을 끌고 있다. 그러나 코드의 질이 충분히 좋다는 것을 담보하기 위하여서는 그러한 신속원형은 어쨌든 습관된 《조기》원형에서보다는 천천히 구성되지 않으면 안된다.

다음으로 신속원형에 대한 관리자측의 영향에 대하여 고찰한다.

１０．７．신속원형에서의 관리문제

신속원형작성에서 나서는 한가지 난점은 어떤 신속원형에 대한 변경에서의 용이성이 의뢰자로 하여금 기능적질이 높은 정식제품판본에 대한 모든 중요한 변경을 요구하도록 부추길수 있다는것이다. 더우기 의뢰자는 제품에 대한 변경을 신속원형에 대한 변경처럼 빨리 실현할수 있게 되기를 바랄수도 있다. 이와 관련되여 제기되는 또 한가지 난점은 비록 신속원형이 필요한 모든것을 할수 있을것 같다 하여도 신속원형이 조작상 품질은 나쁜데 의뢰자는 조작상 품질이 좋은 제품판본을 기대하고 있다는것을 의뢰자에게 설명하여 주는것이다. 신속원형이 리용되기전에 제품개발에 책임 있는 경영자들이 이러한 문제들과 련관된 문제들을 의뢰자와 토론하는것이 아주 중요하다.

모든 새 기술을 도입할 때와 마찬가지로 어떤 개발기업체에서 신속원형작성모형을 도입하기전에 관리자측이 신속원형작성의 우점과 결함을 알아 차리는것이 사활적인 문제로 나선다. 공정하게 말한다면 비록 신속원형작성법에 관한 실례가 강력하다고 하여도 그것은 물론 아직까지 증명되지 않았다. 실례로 자주 인용되는 한가지 실험은 Boehm, Gray, Seewaldt가 진행한 실험인데 그들은 하나의 제품에 대한 7개의 각이한 판본들을 비교하였다[Boehm, Gray, Seewaldt, 1984]. 그중 4개의 판본은 명세서를 작성하였고 3개의 판본은 명세서의 역할을 수행하는 신속원형으로서 원형화되였다. 실험결과는 신속원형작성법과 명세작성법이 거의 같은 성능을 가진 제품들을 주었지만 원형화된 제품이 대략 40% 더 적은 원천코드와 45% 더 적은 노력으로써 개발되였다. 그 원인은 명세작성팀이 아무런 꺼리낌도 없이 명세서안에 경고문들을 첨부하였기때문이다. 다른 한편 신속원형작성자들은 매개 특성을 신속원형안에서 실현하여야 한다는것을 리해하였기때문에 별수없이 본질적이 아닌 모든 기능들을 병합하였다. 평균 40% 적은 원천코드로 하여 원형작성판본이 기능성과 로바스트성이 약간 낮다고 평가되는것은 놀라울것이 없다. 그러나 원형작성판본이 리용 및 학습의 용이성에서 보다 좋다고 평가되는것은 아주 흥미 있다. 결국 명세작성법이 응집도가 보다 좋은 설계와 통합하기 쉬운 쏘프트웨어를 생성하였다. 이 실험에서 한가지 중요한것은 이 실험을 7개의 연구생팀 즉 2명으로 구성된 3개의 팀과 3명으로 구성된 4개의 팀이 진행하였다는것이다. 이 프로젝트는 10주동안 진행되였으며 제품에 대한 유지정비는 진행되지 않았다. 즉 이 실험은 참가자수, 팀규모, 프로젝트규모, 쏘프트웨어개발과정의 견지에서 볼 때 실지제품에 관한 전형으로서 볼수 없다. 그

러므로 Boehm의 결과를 신속원형에 대한 하나의 보증서처럼 리용하는것은 현명하지 못하다. 오히려 이 실험결과를 신속원형작성법의 약점과 우점을 명세작성법과 비교할 때의 지침으로 삼아야 한다. 우에서 언급한 약점외에도 Boehm과 그의 동업자들은 원형작성법으로 만든 제품이 명세작성법으로 만든 제품보다 통합하기 어렵다는것을 밝혀 냈다. 통합의 용이성은 대규모쏘프트웨어제품 특히 C^4I(명령, 조종, 통신, 콤퓨터, 지능)쏘프트웨어에서 중요한 문제이다. 바로 이러한 리유로 하여 신속원형을 명세작성대신에 리용한 그림 10-2의 모형을 리용하지 않고 그것을 요구분석기법으로 리용하는 그림 10-1의 신속원형작성모형을 리용한다.

관리자는 신속원형에 대한 이 두가지 측면을 반드시 고려하여야 한다. 10.5에서는 신속원형을 최종제품으로 전환하는것이 짧은 생각이라는것을 지적하고 있다. 즉 신속원형은 오직 의뢰자의 요구를 정확하게 결정하기 위한 수단으로서만 리용되여야 한다. 또한 일부 경우에 신속원형작성으로 명세작성단계를 대치하고 있다는것이다. 그러나 이것은 결코 설계단계를 대치할수 없다. 개발팀은 신속원형으로부터 얻은 정보와 경험을 훌륭한 설계를 만들기 위한 지침으로 삼을수 있다. 그러나 신속원형은 급히 만들어 지기때문에 훌륭한 신속원형이 훌륭한 설계를 주지 못할수도 있다.

또 한가지 보다 기초적인 론점은 신속원형작성모형의 유지정비는 폭포모형만 운영하는데 습관된 관리자들에 대하여서는 사고방식을 크게 변화시킬것을 필요로 한다는것이다. 폭포모형이 기초하고 있는 개념은 첫번에 정확하게 해야 할것을 수행하는것이다. 틀림없이 폭포모형은 개발팀이 이 목적을 달성하지 못한다면(이 목적은 좀처럼 달성할수 없다.) 수많은 반결합고리들을 결합하게 된다. 그러나 폭포모형개발팀에서 리상적인 점은 개발공정의 매 단계를 오직 한번만 수행한다는것이다. 이와는 반대로 신속원형은 자주 변경되고 그다음 버려 지도록 만들어 진다. 이러한 개념은 관리자들이 일반적으로 습관되여 있는 방법과 정반대이다. 신속원형방법이 여러 차례 반복하면서 목적을 달성하는것은 한번의 실행으로 목적을 달성하는 폭포모형보다 더 형식적일수 있으며 이러한 론리는 관리자가 신속원형작성모형에로 전환하도록 납득시키는데 도움이 될수 있다.

관리자의 관점에서 다른 방법을 필요로 하는 신속원형작성모형의 또 하나의 측면은 의뢰자와 개발자, 특히 신속원형작성팀사이의 호상작용이 중시된다는것이다. 폭포모형에서 의뢰자와 개발자들사이의 호상작용은 개발팀과 의뢰자 그리고 의뢰자와 종업원들사이의 면담들에 의하여 제약된다. 신속원형작성법이 리용될 때에는 신속원형작성팀과 의뢰자팀은 신속원형이 수용될 때까지 거의 련속적으로 호상작용한다.

10. 8. 신속원형의 실험

고돈(Gordon)과 비에만(Bieman)은 신속원형을 리용하고 있는 39건의 공개 및 비공개 실례연구들을 분석하였다[Gordon and Bieman, 1995]. 이 가운데서 33건은 성공으로 인정되고 3건은 실패, 3건은 평가되지 않았다. Gordon과 Bieman은 성공하지 못한 실례연구들은 거의 공개되지 않았다는것을 지적하였다. 결론적으로 그들은 자기들이 진행한 분석이 신속원형작성으로 인한 높은 성공률을 반영하고 있다고 주장하지 못하였다. 오히려 그들

의 론문은 신속원형이 성공적으로 리용된 여러가지 성공적쏘프트웨어프로젝트들에 대한 보고서로 된다.

공개된바와 같이 실례연구들은 매개의 흥미 있는 론점들을 전혀 반영하지 못하고 있다. 실례로 사용의 용이성에 관한 론평은 오직 17건의 실례연구에서만 제시되였다. 신속원형작성으로 인한 정식제품의 사용용이성에서의 개선은 이 17건의 실례모두에서 보고되고 있다. 기타 22건의 실례연구들은 제품을 사용하기 어렵다는것을 주장하지 못하고 있다. 그러므로 비록 자료가 불완전하고 전문적인 화제들이 명백하게 언급되지 않았다고 하여도 일련의 결론을 내릴수 있다.

사용자관여에 대하여 론평한 모든 실례연구들은 열성적인 관계자들은 이미 개발공정에 참가하고 있다는것을 보고하고 있다. 또한 신속원형들은 의뢰자의 요구를 만족시킨다는것이 밝혀 졌다. 기타 론쟁에서는 또 다른 견해들도 있다. 실례로 4건의 연구들은 신속원형이 제품들이 보다 적게 배포되게 하였다고 보고하고 있으며 한건은 원천코드규모가 증대되였다는것을 보고하고 있다. 많은 실례연구들에서 신속원형작성에서 몇가지 불필요한 특성들이 실현되였다는것을 언급하고 있지만 두건은 이와 같은 특성들이 증가하였다는것을 보고하고 있다.

언어선택은 신속원형작성에서 더욱 중요한 문제로 제기된다. 39건의 실례연구에서 총 26개의 각이한 언어들이 리용되였다. 가장 많이 쓰인 언어는 Lisp(4건)와 Smalltalk(3건)였다.

결과들에서 가장 중점으로 된것은 신속원형을 버려야 하는가에 관한 문제이다. 수많은 쏘프트웨어개발과정들이 리용되였기때문에 이 론쟁에 대하여 명확한 결론을 짓기는 어렵다. 일부 실례들에서 신속원형은 통채로 최종제품으로 전환될 때까지 성과적으로 완성되였다. 또 다른 실례들에서는 신속원형의 일부만이 보류되고 나머지는 버리였다. 하나의 실례에서 설계단계가 신속원형의 반복을 요구하였다. 일부 실례연구들에서는 관리자측이 사전에 신속원형이 보류되고 완성될것이라는것을 규정하였다. 실례연구중에서 22건은 신속원형전체 혹은 부분을 보류완성해야 한다는것을 특별히 권고하고 있으며 8건은 신속원형을 버린다는것을 주장하였다.

신속원형을 보류하는것은 규모가 큰 쏘프트웨어제품에서 특별히 중요하다는것이 밝혀 졌다. 39건의 실례연구에서 7건은 규모가 방대한데(10만행이상의 코드) 7건모두가 신속원형전체 또는 부분을 보류하고 있다. 모든 제품들에서 신속원형을 버려야 한다고 믿는것이 일반적인 견해이다. 이후 이 분야에 대하여 연구할 필요가 있다. 특히 10.5에서 서술된 변형된 신속원형작성모형을 리용하여야 하는가를 아는것이 중요하다. 즉 신속원형이 작성되기전에 원형의 부분들이 최종제품에서 활용될수 있는가 그리고 개발자가 보류하려고 하는 부분들이 설계와 코드검토를 거쳐야 하는가를 관리자측에서 결정하여야 한다.

이러한 연구로부터 일련의 림시적인 결론을 내릴수 있다.

첫째로, 신속원형작성이 성공적인 쏘프트웨어개발에로 이끌어 갈수 있는 실행가능한 기법이라는것이다. 동시에 수많은 위험가능성도 있다. 만일 신속원형이 보류되면 결과적인 제품이 잘못 설계되여 유지정비가 어렵게 되고 불완전하게 수행될 위험성이 있다. 이러한 위험성은 아마도 제품의 규모에 비례하여 커지는것 같다. 그러나 이러한 위험성은 감소시킬수도 있다.

그 한가지 방법은 10.5에서 서술한 변형된 신속원형을 리용하는것이다. 실례연구에서의 보고와 의도적으로 조종한 실험사이에는 차이가 있다. 그러나 쏘프트웨어공학에서 조종실험은 매우 어려우며 한가지 다른 방법은 다른 사람들의 경험을 배우는것이라는것을 명심해야 한다.

10. 9. 요구사항도출 및 분석을 위한 기법

1차적인 요구사항도출기법은 면담을 진행하는것이다(10.1.1). 이것을 례증하기 위하여 요구분석팀이 프로젝트의 첫 시작에서 하나의 신속원형을 작성한다고 가정하자. 그러나 신속원형의 첫 반복이 작성되기전에 개발팀과 관련된 의뢰자와의 면담이 철저히 진행되면 될수록 첫 반복이 의뢰자의 요구를 정확하게 반영할 가능성은 더 커진다. 다른 한편 신속원형을 단순히 모아 놓으면 긴 세련과정을 수행해나가는데서 시간이 적게 걸리거나 비용도 효과적이다. 왜냐하면 명백히 신속원형작성은 요구분석기법이지 요구도출기법이 아니기때문이다.

만일 폼(*form*)기법이 리용된다면 또한 면담을 진행하는것이 필요하다(10.1.3). 폼, 보고서, 일감서술을 주의 깊게 음미해 보는것은 매우 중요하다. 그러나 이러한 고찰로부터 하나하나 수집한 정보의 정확성을 확인하기 위한 유일한 방도는 련관된 의뢰자측 성원들과 면담하는것이다. 결국 이 절의 시작에서 주장한바와 같이 면담은 사실상 1차적인 요구도출기법이다. 한편 10.2에서 지적된바와 같이 신속원형작성은 가장 정확한 요구분석기법이다. 신속원형작성은 제품의 작업모형을 구성할수 있게 하여 기타 기법들보다 의뢰자의 실지요구를 더 잘 만족시키는것 같다. 신속원형을 리용하지 않으면 요구분석단계가 부적당하게 수행되며 의뢰자의 요구가 목적한 제품을 완전히 만족시키지 못할 중대한 위험성이 발생한다.

10. 10. 요구사항확정단계에서의 시험

비록 요구사항확정단계의 목적이 의뢰자의 실지요구를 확정하는것이라고 하여도 일반적으로 의뢰자는 목적하는 제품의 1차사용자로는 되지 않는다. 그렇기때문에 사용자에게 신속원형을 실험하며 의뢰자가 찬성하는 경우에 쏘프트웨어제품의 배포판에서 실현될 변경을 제기할 기회를 주는것이 매우 중요하다.

그렇기때문에 신속원형작성단계에서 쏘프트웨어품질보증그룹의 임무는 의뢰기업체의 련관된 개별적사람들이 신속원형과 작용하는 기회를 가지며 그들의 제의가 의뢰자 또는 사용자의 제의를 분석할 책임이 있는 의뢰자측에 정확히 전달된다는것을 확인하는것이다.

10.2에서 언급된바와 같이 앞으로 수행할 단계에서의 시험견지로부터 요구사항확정은 본질적으로 추적가능해야 한다. 이것을 위하여 요구문들에 번호를 달아 SQA가 이후 단계들에서 그것들을 추적할수 있게 하는것이 필요하다. 번호달기는 요구서의 매 항목들을 실현하는 명령묶음들이 린접하여 있는 주석의 형식으로서 신속원형에 표현되여야 한다.

1 0. 1 1. 요구사항확정단계를 위한 CASE도구

해석형언어들은 일반적으로 신속원형작성을 위한 훌륭한 매체로서 복무한다. 해석형언어들이 콤파일되거나 링크될 필요가 없기때문에 신속원형의 개발을 보다 빠르게 한다. 또한 의뢰자가 제기하는 작은 변경요구도 신속원형이 현시되는 기간에 자주 실현될수 있다. 해석형언어들은 일반적으로 콤파일형언어보다도 실행에서 덜 효과적이며 유지정비에서 제기되는 난점들도 일부 해석형언어들과 관련되여 있다. 그러나 신속원형내에서는 이러한 론쟁이 아무런 관계도 없다. 요점은 신속원형을 빨리 종합하고 그다음 버리는것인데 대부분의 해석형언어들은 이 목적을 실현하는데서 리상적이다. 신속원형작성언어와 련관되여 있는 CASE도구들을 리용하면 더 좋은 효과들을 얻을수 있다. 실례로 Smalltalk는 여러가지 수단으로서 사용자를 돕고 신속원형작성과정을 촉진하는 환경을 가지고 있다. Interlisp환경은 Lisp프로그람작성자들을 지원하는 이와 류사한 제품이다. Java는 이식성이 좋은 해석형언어이다. 많은 Java개발환경들이 현재 보급되고 있는데 앞으로 더욱 기대가 클것으로 보인다. 앞에서 언급한것처럼 Perl은 신속원형을 작성하기 위한 보편적인 프로그람작성언어이다. HTML은 광범히 리용되고 있는 또 한가지 신속원형작성언어이다. 만일 신속원형을 버려야 한다면(10.7에서 서술된바와 같이) HTML은 보다 좋은 우월성을 나타낸다. 배포제품을 HTML로 작성한다는것은 거의 생각조차 할수 없는 일이므로 HTML의 리용은 사실상 신속원형이 버려 질것이라는것을 담보하고 있다.

최근년간에 Oracle, PowerBuilder, DB2와 같은 4세대언어들이 신속원형작성에 리용되였다(4GL은 14.2에서 론의된다). 여기에는 여러가지 원인이 있다. 첫째로 4세대언어의 설계목적이 보다 적은 명령들로서 Java, Ada, C++와 같은 3세대언어들과 같은 기능을 달성하는것이다. 결국 신속원형은 4GL을 리용할 때 아마도 더 빨리 배포될것이다. 둘째로 대다수 4GL들은 해석형이다. 이것은 이 절의 서두에서 서술한것처럼 신속원형작성을 촉진시킨다. 셋째로 대다수 4GL은 강력한 CASE도구들로 지원된다. 결국 신속원형작성과정을 더욱 촉진시킨다.

다른 한편 신속원형작성에서 4GL의 리용은 고유한 결함을 가질수 있다. 4GL이 매장된 CASE환경은 자주 완전한 쏘프트웨어개발공정에 리용될 보다 큰 도구모임의 일부분으로 된다. 4GL공급자가 일반적으로 권고하는 쏘프트웨어개발공정은 신속원형을 작성하고 그다음 그것을 최종적인 제품으로 될 때까지 성과적으로 세련시키는것이다. 결국 공급자는 흔히 쏘프트웨어개발공정이 구성 및 수정모형으로 퇴보하는가에 관심을 돌리지 않는다. 공교롭게도 대다수 4GL공급자들의 유일한 목적은 개발팀에서 자기들의 단일한 제품 즉 어떤 프로젝트의 모든 국면을 조종할수 있는 단일한 CASE환경을 구입한다는것을 확인하는것이다.

쏘프트웨어개발관리자가 쏘프트웨어개발공정의 모든 관계를 지원할 어떤 CASE환경을 구입하기 위해 수만딸라를 지출하였다고 하자. 이제 신속원형들이 버려 질수 있다는것을 확증하기 위하여 신속원형작성에 리용될 다른 언어로 된 개발프로그람을 사는데 돈을 더 지불하도록 경영자를 납득시키는것은 쉬운 일이 아니다. 그것은 경영자가 자기가 이미 새로운 개발환경의 한 부분으로 되는 아주 좋은 신속원형작성도구를 구입하였는데 불필요한 추가적인 신속원형작성수단을 사는데 돈을 들이라고 요구한다고 반박할수 있기때

문이다. 그러므로 경영자에게 신속원형작성을 위한 추가적인 수단이나 작업대(*workbench*)를 사는데 드는 비용이 신속원형을 생산품질이 좋은 쏘프트웨어로 전환하고 그다음 그 제품을 유지정비하려고 할 때 드는 잠재적인 거대한 비용에 비해 작다는것을 보여 주어야 한다.

１０．１２．요구사항확정단계에서의 척도

요구사항확정단계의 한가지 중요한 특성은 요구사항확성팀이 의뢰자의 실제요구를 얼마나 빨리 결정하는가 하는것이다. 그러므로 요구사항확정단계에서의 한가지 유용한 척도는 요구휘발성에 대한 측도이다. 요구사항확정단계에서 요구가 얼마나 자주 변경되는가를 기록하는것은 관리자측에게 요구사항확정팀이 제품의 실제적요구에 대한 집중률을 결정하기 위한 방법을 제공하여 준다. 이와 같은 척도는 그것이 면담이나 대본작성과 같은 임의의 요구도출기법에 적용될수 있다는 우월성을 가지고 있다.

요구사항확정팀이 자기의 과제를 얼마나 잘 수행하고 있는가를 재는 또 한가지 측도는 쏘프트웨어개발의 나머지 단계들에서 변경되는 요구의 개수이다. 만일 명세작성, 설계 그리고 그이후 단계들에서 많은 요구들이 변경되야 한다면 팀이 요구사항확정단계를 수행하는 방법을 철저히 분석하여야 한다는것은 명백한 사실이다. 이러한 척도는 이 장에서 서술되는 모든 요구도출기법들에 적용할수 있다.

신속원형이 리용될 때 유용한 한가지 척도는 의뢰자와 사용자가 신속원형을 실험할 때 매 신속원형특성들이 시도되는 회수이다. 실례로 만일 매 사용자가 차림표에서 화면 J를 적어도 한번 선택하였으나 화면 B는 누구도 선택하지 않았다면 개발팀은 의뢰자에게 이 두 화면에 대하여 질문하여야 한다. 특히 개발자들은 화면 J가 그에 대한 실행시간이 최소로 되도록 설계할 정도로 중요한가 또 화면 B가 제품에 포함될 필요가 있는가에 대하여 알아야 한다. 만일 그렇다면 적어도 한명의 사용자는 화면 B를 실험하여야 한다. 모든 화면들이 사용자의 요구를 충족시켜야 한다는것은 사활적인 문제이다.

１０．１３．객체지향요구가 존재하는가

이 장에는 객체지향파라다임에 속하는것이 전혀 없다. 이것은 무엇때문인가고 묻는것은 응당하다. 이 책의 제목이 객체지향 및 고전쏘프트웨어공학인데 객체지향요구에 관한 자료가 없다는것은 심중한 생략처럼 생각될수도 있다.

이에 대한 대답은 《객체지향요구》라는것은 전혀 없으며 또 없어야 한다는것이다. 요구사항확정단계의 목적은 의뢰자의 요구를 결정하는것이다. 즉 목적하는 체계의 기능은 무엇으로 되여야 하는가 하는것이다. 요구사항확정단계에서는 체계가 어떻게 작성되여야 하는가에 대하여서는 아무것도 하지 않는다. 고전적사용자안내서 또는 객체지향사용자안내서를 언급하는것이 의미없는것과 마찬가지로 요구사항확정단계에서 고전적파라다임이나 객체지향파라다임을 언급하는것은 의미가 없다. 사용자안내서는 쏘프트웨어제품을 실행할 때 사용자가 수행하여야 할 단계들을 서술하고 있지 제품이 어떻게 만들어졌는가에 대하여서는 아무것도 서술하고 있지 않다. 이와 같은 방식으로 요구사항확정단

계에서는 제품이 무엇을 하기로 되여 있는가를 서술하며 그 제품이 작성되는 방법은 그 안에 들어 가지 않는다.

객체지향요구와 같은 말은 없다는것을 범주적으로 서술하면서도 그러나 객체지향파라다임이 사실상 요구사항확정단계에 들어 갈수 있는 한가지 방법이 있다는것을 언급하지 않을수 없다. 이 장에서 강하게 권고한바와 같이 어떤 특정한 프로젝트에 대한 요구사항확정단계는 신속원형작성을 포함한다고 가정하자. 이 신속원형을 위한 코드작성의 결과로서 개발팀은 목적하는 쏘프트웨어에서 무엇이 클라스를 구성할수 있는가에 관한 생각을 비롯하여 문제령역에 대하여 간파하게 될것이다. 즉 신속원형이 C 또는 Lisp와 같은 고전적인 언어로 작성되였다고 하여도 개발팀은 구성될 제품에 대한 기본구성블로크들을 파악하게 될것이다. 그러면 다음 단계 즉 객체지향분석(OOA)에서 이 정보들은 OOA의 첫단계에서 클라스들을 추출하는데 도움이 될수 있다(12.2). 이와 달리 객체지향파라다임은 요구사항확정단계에서 아무런 역할도 놀지 못한다.

10. 14. 항공음식전문회사 실례연구: 요구사항확정단계

많은 항공려행자들은 자기들의 음식요구를 가지고 있으며 대다수의 항공회사들은 그들의 요구를 만족시키는 음식을 공급하려고 노력한다. 실례로 대부분의 항공회사들은 남새료리, 해산물, 정결한 식료품은 물론 당뇨병환자용음식, 저지방, 저콜레스트린, 저단백, 저카로리, 저염식사들을 봉사하는데 충분한 주의를 돌린다. 어린이들의 식사도 아무때나 주문할수 있다. 일부 항공회사들은 무유당식사를 공급할것이다. 비행기안내원들에게는 특별식사를 요구하는 승객들의 명단과 그들의 좌석번호가 제공된다. 그러면 특별식사가 규정식사와 같은 시간에 봉사된다.

거의 모든 기업결재에서와 마찬가지로 이러한 특별식사를 제공할 때 거래가 진행된다. 려객인원수를 증가시키고 려객들에게 친절히 봉사하는데서 리익이 돌아 온다. 결국 어떤 항공회사가 특별식사봉사를 하지 않게 되면 특별식사봉사를 하는 항공회사보다 승객들을 잃게 될것이다. 다른 한편 특별식사를 제공하는데 비용이 든다.

1. 특별식사에 들어 가는 재료들은 규정식사용재료들보다 값이 비쌀수 있다.
2. 특별식사는 상대적으로 적게 봉사되기때문에 대량구입에 의한 저축은 있을수 없다.
3. 일부 항공회사들은 자기들의 주방에서 특별식사를 준비한다. 그대신 그들은 외부료리공급원과 계약을 맺고 특별식사를 공급하는데 이것은 비용을 더욱 증대시킨다.
4. 매개 특별식사는 중심주방으로부터 비행이 시작되는 비행장까지 수송되여야 한다. 거래비 및 사무처리비용이 이 과정에 포함된다.
5. 흔히 특별식사는 예정된 접수자가 소비하지 않게 된다. 실례로 만일 어떤 려객이 마지막순간에 자기의 려행계획을 바꾸거나 비행기가 늦게 도착하게 되면 식사는 그것을 주문한 원래 비행기에 실려 있게 된다.
6. 비록 특별식사는 미리 주문되였다고 하여도 사람들의 실수로 하여 때때로 해당한 비행기에 정확히 실려 지지 않게 된다.

총적으로 대다수 항공회사들은 특별식사를 봉사하는데서 얻는 리득이 비용에 비하여 훨씬 가치가 있다는것을 알게 되였다. 항공음식전문회사는 자기들이 항공봉사하는 고도로 표준화된 음식들을 늘 자랑하여 왔는데 그 경쟁자들은 현재 기껏하여 맛이 없는 작은 땅콩구럭을 공급하고 있다.

지금까지 항공음식전문회사의 수익성은 비행기안에서 항공전문식사를 하는데 좀 비싼 값을 물어도 무방해 하는 승객들에 대한 안정된 공급에 의하여 담보되여 왔다. 그러나 최근에 항공음식전문회사의 수입한계는 축소되였으며 회사관리자측은 경비를 줄일 방도를 찾고 있다. 특별식사를 공급하는데 드는 높은 비용은 그것을 주문한 승객에게 극히 적은 량의 특별식사가 차례진다는 일화와 결합되여 특별식사를 항공음식전문회사의 통계원을 비난하는 요인으로 되게 하였다.

그러나 그 어떤 조치를 취하기에 앞서 항공음식전문회사 관리자측은 특별식사계획의 성공과 실패에 관련된 믿음직한 자료를 얻으려고 한다. 또한 특별식사는 외부공급원에 의하여 공급되기때문에 일부 항공회사관리자측은 항공음식전문회사의 주방에서 품질조종으로 인하여 이 식사들이 규정된 항공음식전문회사음식의 높은 기준을 만족시킬수 없지 않는가고 의심한다. 그렇기때문에 승객의 만족성검사는 특별식사의 품질인식을 결정하는데로 이어 져야 한다. 또한 항공음식전문회사 관리자측은 특별식사가 비행기에 정확히 실리지 못한 비률을 알려고 한다.

특별식사를 봉사하는 기타 대다수의 항공회사들과 마찬가지로 작성된 비행시간 24h 전에 항공음식예약체계의 자료기지는 특별식사에 대한 요구를 주사한다. 공급원들이 특별식사가 련관된 비행기에 제때에 운반된다는것을 확인할수 있도록 보고서가 작성된다. 그다음 모든것이 운반된 직후에 비행기안내원을 위한 보고서가 비행기에 탑재된 콤퓨터에서 인쇄된다. 이 보고서에는 예약식별자, 이름, 특별식사를 주문한 비행기에 오른 모든 승객들의 좌석번호, 식사류형 등이 포함된다(승객들은 비행기에서 식사를 하기전에 어떤 형식으로 자기의 신분을 확인시켜야 한다. 결국 비행기의 콤퓨터는 이 보고서를 인쇄하는데 필요한 정보를 얻게 된다.).

특별식사에 관하여 관리자측이 요구하는 정보를 얻기 위하여서는 항공음식전문회사 쏘프트웨어에 일정한 변경을 가해야 한다.

첫째로, 특별식사가 주문될 때마다 다음과 같은 정보를 기록하여야 한다. 즉 예약식별자, 비행기번호, 날자, 시간, 승객이름과 주소, 주문한 특별식사류형을 기록한다. 이 정보는 새로 작성한 프로그람이 특별식사자료를 분석하는데 리용될것이다. 둘째로, 비행기 콤퓨터에서 인쇄되는 특별식사목록은 세로 렬 지은 선택직사각형들을 포함하게 될것이다. 비행기안내원들은 만일 관심하는 식사가 비행기에 실렸다면 해당한 직사각형이 그늘지게 표시한다.

비행이 끝난후 이 목록이 검토된다. 만일 해당한 직사각형이 그늘지게 표식되였다면 그 승객은 비행기에 탔다는것을 말하며 예약식별자를 포함하고 있는 우편엽서가 승객에게로 보내지며 그에게 1부터 5등급까지의 식사를 평가하도록 요구한다.

우편엽서가 되돌아 올 때 그것을 또 조사하고 후에 보고서로 작성할 목적으로 예약식별자와 응답을 기록하여 놓는다.

결국 특별식사분석프로그람에는 3개의 독립적인 순차적단계가 있게 된다. 즉 출발

24h전에 만들어 지는 추가기록, 조사목록, 조사우편엽서가 있다.

기록자료요소들의 형식은 다음과 같다[선택마당들은 중괄호안에 넣었다.].

예약식별자(6개의 대문자)
비행기번호(3개의 수자, 오른쪽으로 정돈되며 령이 채워 진다.)
비행날자(9개의 문자: 2개의 수자로 날자표시, 3개의 대문자로 달표시, 4개의 수자로 년표시)
좌석번호(3개의 수자는 오른쪽으로 정돈되며 령이 채워 진다. 그앞에 하나의 문자가 있다)
승객이름
    성(15개 문자까지 허용)
    [중간의 첫 문자(1개 문자)]
    이름(15개 문자까지 허용)
    [뒤붙이(5개 문자까지 허용)]
승객주소
    주소의 첫행(25개 문자까지 허용)
    [주소의 두번째 행(25개 문자까지 허용)]
    도시(14개 문자까지 허용)
    [주, 구역, 지구(14개 문자까지 허용)]
    [우편대호(10개 문자까지 허용되며 수자, -, 공백을 리용)]
    나라(20개 문자까지 허용)
특별식사류형(어린이용, 당뇨병환자용, 유태인용, 회교도용, 무유당, 저카로리, 저콜레스크린, 저지방, 저단백, 저염, 해산물, 남새료리)
승객이 비행기에 올랐는가?(1개 문자)
특별식사를 실었는가?(1개 문자)
식사품질(1∼5)

새 쏘프트웨어는 자료기지로부터 정보를 입력하고 다음 4가지 보고서가운데서 사용자가 선택할수 있게 하여야 한다. 매 보고서에서 그 보고서의 시작과 끝 날자는 사용자로부터 얻어 져야 한다.

1. 매 특별식사류형과 특별식사프로그람에 대하여 다음과 같은 내용을 포함하고 있는 보고서
    특별식사가 규정된대로 오른 비률
    특별식사를 주문한 승객이 비행기에 오른 비률
    특별식사를 주문하였지만 그 식사가 오르지 않은 승객들의 비률

려객관계부서는 다음과 같은 보고서를 요구한다.

2. 보고서작성기간내에서 특별식사가 한번이상 오르지 않은 승객들의

이름과 주소 및 이 사건의 발생날자

3. 특별식사질이 최고급(5급)보다 낮다고 생각하는 승객들의 이름과 주소 및 이 사건의 발생날자 그리고 식사류형

저염식사를 공급하는 외부공급원은 자기 자신의 품질조종프로그람을 가지고 있어야 한다. 그를 돕기 위하여 다음과 같은 보고서가 또 필요하다.

4. 봉사한 매 저염식사에 대하여 비행기번호와 날자, 식사의 품질급수

항공음식전문회사관리자측은 이러한 추가적인 기능을 병합함으로써 자기들의 현존 콤퓨터체계가 변경될 위험성에 대하여 크게 우려한다. 또한 관리자측은 새로운 제품이 완성될 때까지 내키지는 않지만 여러가지 검사기들을 구입한다. 따라서 건반으로 구동되는 자립형제품이 만들어 져야 할것 같다. 즉 이 제품에서는 예약방식을 입력하고 승객을 검사하며 인쇄된 특별식사를 조사한다(이것은 건반입력으로 모의되게 된다.). 그리고 우편엽서를 조사(이것도 건반입력으로 진행)하고 여러가지 보고서를 만들어 낸다. 때 이른 제품완성에 대처하기 위한 앞으로의 예방책이 현존 콤퓨터체계를 변경시킬 때 항공음식전문회사관리자측은 외부쏘프트웨어기업체를 고용하여 자립형제품을 만들것을 결정하게 된다.

10.15. 항공음식전문회사실례연구: 신속원형작성

항공음식전문회사관리자측이 자기들의 요구를 정확히 결정하기 위하여 직원들과 심중하게 면담을 하였음에도 불구하고 최종제품이 의뢰자의 실제적요구를 정말 만족시킨다는것을 확인하기 위한 유일한 방도는 신속원형을 작성하는것이다. 항공음식전문회사쏘프트웨어제품에 리용될 콤퓨터에 적합한 언어들은 C, C++와 Java이다. 만일 제품이 C++로 실현되게 되면 해석형언어인 Java로 신속원형을 작성하는것이 적합하다. 그러나 구성 및 수정문제가 발생하는데도 불구하고 실현팀이 Java신속원형을 C++로 전환하고 보충적인 기능을 추가할것이라는 위험성이 존재한다(3.1). 다른 한편 만일 제품이 Java로 실현된다면 C도 C++도 특별히 좋은 대용물로는 되지 않는다. 그 두개 언어가운데서 C가 아마도 더 좋을수 있다. 왜냐하면 그렇게 되면 신속원형이 완전히 Java로 작성되여야 하기때문이다. 만일 C++를 리용한다면 실현팀은 생산성이 좋은 판본을 처음부터 실현하지 않고 신속원형을 Java로 변환하고 그것을 변경시킬수 있다.

C와 Java신속원형들은 www.mhhe.com/engs/compsci/schach에서 리용할수 있다. 이 원천코드들은 일반적으로 명백하다. 그러나 3가지 측면은 여기서 반드시 주의해야 한다.

C신속원형의 한 부분을 그림 10-4에서 보여 주었다. 그리고 그에 대응하는 Java신속원형부분은 그림 10-5에서 보여 주었다. 먼저 그림 10-4 또는 그림 10-5에서 보여 준바와 같이 10명까지의 승객레코드가 하나의 배렬안에 기억된다. 정식제품에서 승객수를 하나의 변수로 하는 자료구조가 요구된다. 실례로 파일 또는 동적자료구조가 리용될수 있

다. 그러나 신속원형에서 실현하기 쉽고 빠르기때문에 배렬이 리용되며 신속원형은 몇개의 승객레코드로써 검사될수 있다. 류사하게 20개까지의 비행기록이 다른 배렬에 기억된다. 이것도 그림 10-4와 그림 10-5에 반영되여 있다.

신속원형작성에 관한 두번째 중요한 측면은 그것이 미완성품이라는것이다. 실례로 6개의 보고서가운데서 저녁식사에 대한 보고서와 비행기에 오른 식사에 대한 보고서 두개만이 실현되였다. 이것은 다른 4개의 보고서가 이 두개의 보고서와 류사하기때문이다. 이 4개의 보고서는 스터브(대용체; *stub*) 즉 몸체부는 없이 하나의 대면부로 이루어 지는 무의미부분프로그람으로서 작성된다. 이 부분프로그람들은 호출될 때 통보문을 현시한다. 이에 대한 한가지 실례를 그림 10-4와 그림 10-5에서 보여 주었다. 이 부분프로그람의 몸체부를 생략하는것은 신속원형의 기능을 크게 손상시킴이 없이 그 개발을 촉진시킨다.

```
#define    NUM_PASSENGER_RECORDS    10
#define    NUM_FLIGHT_RECORDS       20
…
struct     passenger_type  passenger_records[NUM_PASSENGER_RECORDS];
struct     flight_record_type   flight_records[NUM_FLIGHT_RECORDS];
…
printf ("\t\t     24HOUR   CATERER   LIST\n\n");
printf ("\t\t This report is not inplemented in the rapid prototype\n\n\n");
printf ("      Press <ENTER> to return to the menu");
```

그림 10-4. 항공음식전문회사제품을 위한 C신속원형부분

```
public static final int    NUM_PASSENGER_RECORDS  = 10;
public static final int    NUM_FLIGHT_RECORDS     = 20;
…
public static CPassenger   passengers[] =
                           new CPassenger [NUM_PASSENGER_RECORDS];
public static CFlightRecord   fltRecs[]=
                           new CFlightRecord [NUM_FLIGHT_RECORDS];
…
System.out.println ("        24 HOUR CATERER LIST \n\n");
System.out.println ("  This report is not implemented in the rapid prototype\n\n\n");
System.out.println ("  Press <ENTER> to return to the menu…");
```

그림 10-5. 항공음식전문회사제품을 위한 신속원형부분

마지막으로 신속원형의 사용자대면부는 차림표구동식으로 작성된다. 이것은 그림 10-4와 그림 10-5의 마지막행에 암시되여 있다. 차림표구동식대면부는 도형사용자대면부처럼 세련되여 있지는 않지만 두가지 우점을 가지고 있다. 첫째로, 신속원형을 작성할 때 속도가 빠르다는것이다. 강력한 GUI발생기를 리용하지 않는 한 일반적으로 단순한 차림표구동식사용자대면부를 코드작성하는것이 더 빠르다. 둘째로, GUI는 흔히 하드웨어와

조작체계에 의존한다. 만일 정식제품이 신속원형이 실행되는것과 크게 차이나는 하드웨어와 조작체계상에서 실현된다면 GUI는 처음부터 다시 실현되여야 할수도 있는데 여기에는 추가적인 비용과 시간이 소비된다. 이 두가지 원인으로 하여 신속원형은 단순한 본문적사용자대면부형식으로 작성하는것이 더 좋다. 보다 중요한 사용자대면부를 작성하기 위한 결심은 생명주기의 뒤단계들에서 작성될수 있다.

10. 16. 요구사항확정단계의 난관

쏘프트웨어개발공정의 다른 매 단계들과 마찬가지로 요구사항확정단계에서도 잠재적인 문제점들과 함정들이 존재한다. 첫째로, 목적하는 제품의 잠재적사용자들과 개발공정의 시작에서부터 성심성의로 협동하는것이 매우 중요하다.

개별적종업원들은 콤퓨터가 자기들의 일자리를 빼앗게 될가봐 두려워 하면서 콤퓨터화를 실현하는데 흔히 압박감을 느낀다. 이른바 콤퓨터시대의 직접적인 또는 간접적인 결과로써 일어 난 많은 나라들에서의 경제의 불균형적인 성장은 콤퓨터화의 결과로 일자리를 잃는 개별적사람들에게 미치는 부정적충격을 보상할 방도가 없다.

요구분석팀의 모든 성원들은 자기들이 대상하는 의뢰자측 성원들이 목적하는 제품이 그들의 일자리에 주는 잠재적인 충격에 대하여 깊이 우려할수 있다는것을 생각해야 한다. 최악의 경우에 종업원들은 제품이 의뢰자의 요구를 만족시키지 못한다는것을 확인시키고 이것으로 자기들의 일자리를 보호하기 위하여 일부러 오해하기 쉽거나 틀리는 정보들을 제공할수도 있다. 그러나 이러한 고의적인 태업은 없다고 하여도 일부 의뢰자측 성원들은 단순히 콤퓨터화로 인하여 받는 압박감때문에 잘 협조하지 않을수 있다. 요구사항확정단계에서의 또 한가지 난관은 협상가능성이다. 실례로 의뢰자가 원하는것을 등급별로 규정하는것이 중요하다. 놀랄것 없이 거의 대부분의 의뢰자들은 필요하다고 생각되는 모든것을 수행할수 있는 쏘프트웨어제품을 가지기를 즐겨 한다.

이와 같은 제품을 만드는데는 용납할수 없을 정도로 긴 시간이 걸리며 의뢰자가 타당하다고 생각하는 비용보다도 훨씬 더 비싼 값이 든다. 그러므로 의뢰자들이 바라는것보다 적은(때때로 훨씬 더 적은) 요구를 받아 들이도록 의뢰자들을 설득하는것이 필요하다.

론의하는 매 요구들의 비용과 리익을 계산하는것(5.2)이 이러한 고려에서 도움을 줄수 있다.

또 한가지 실례의 필요한 협상기교는 목적한 제품의 기능성에 관하여 경영자들사이의 타협에 도달하는 능력이다. 실례로 교활한 경영자는 현재는 다른 경영자의 책임인 어떤 업무기능을 자기의 책임령역에 병합시킴으로써만 실현될수 있는 어떤 요구를 포함시키는것으로써 자기의 권한을 확장하려고 기도할수도 있다. 놀랄것 없이 다른 경영자는 이것을 발견하는 경우 강하게 대항할것이다. 이 경우 요구작성팀은 두 경영자를 진정시켜 론쟁을 해결하여야 한다.

요구사항확정단계의 세번째 난관은 대다수 기업체들에서 요구사항확정팀이 도출하려고 하는 정보를 가지고 있는 개별적사람들에게 있어서 깊이 있는 토론을 진행하기 위하여 만날 시간이 모자라는것이다. 이런 경우가 발생하면 요구사항확정팀은 어느것이 더 중요한가를 결정해야 할 의뢰자에게 개별적사람들의 과제책임이나 구성될 쏘프트웨어제

품을 알려 주어야 한다. 그리고 만일 의뢰자가 그 쏘프트웨어제품이 처음 나오는것이라고 주장하는데 실패하면 개발자는 거의 실패할 운명에 직면한 프로젝트로부터 물러서는것외에 다른 대책이 없을수 있다.

마지막으로 유연성과 객체성은 요구도출에서 본질적인 문제이다. 요구사항확정팀성원들이 아무런 사전견해가 없이 면담에 림하는것이 사활적인 문제이다. 특히 면담자는 앞선 면담결과에 기초하여 요구들에 대한 가정을 절대로 하지 말아야 한다. 그러면 이 가정의 틀안에서 다음면담이 진행되게 된다. 그대신에 앞선 면담에서 하나하나 얻어 낸 모든 정보들을 의식적으로 덮어 두고 매 면담을 공평한 방법으로 이끌어 나가야 한다. 요구와 관련하여 너무 이른 가정을 하는것은 위험하다. 작성된 쏘프트웨어를 고려하여 요구사항확정단계에서 그 어떤 가정을 하는것은 재난을 가져 올수 있다.

요구사항확정단계의 진행과정

요구도출

- 의뢰자측 성원들과 그들의 요구를 결정한다.

요구분석

- 예비적인 요구문서를 작성한다.
- 요구문서를 세련시킨다.
- 목적하는 제품의 중요기능들을 병합한 신속원형을 작성한다.
- 신속원형을 실험한 의뢰자측 성원들로부터 반결합정보에 따라 신속원형을 변경한다.

요 약

이 장은 요구사항도출기법의 서술로 시작하며(10.1) 뒤이어 요구사항분석의 륜곽을 보여 주는데(10.2) 그것은 우의 란에 종합되여 있다.

신속원형작성에 관한 자세한 정의를 주었다(10.3). 사용자대면부를 설계할 때 인간적 인자들을 고려하는것의 중요성이 10.4에서 론의되였다. 신속원형을 명세작성법으로 리용하는 방법은 그리 좋은 방법이 아니라는것을 10.5에서 보여 준다. 10.6에서는 신속원형의 재리용을 서술한다. 신속원형작성에 대한 관리자측의 영향은 10.7에서 론의했다. 신속원형의 시험이 10.8에서 론의되고 요구사항도출을 위한 기법들이 10.9에서 비교되며 요구사항분석을 위한 기법들이 제시된다. 그다음 신속원형을 검사하기 위한 방법들(10.10), 요구사항확정단계를 위한 CASE도구들(10.11), 요구사항확정단계를 위한 척도들(10.12)이 서술되였다. 이 장은 객체지향요구사항확정단계가 존재하는가에 관한 론의(10.13), 신속원형(10.15)을 비롯하여 항공음식전문프로그람에 관한 요구사항확정단계(10.14), 요구사항확정단계의 난관(10.16)으로 결속된다.

보 충

요구사항분석에 대한 도서들에는 문헌 [Gause and Weinberg, 1990; Davis, 1993; and Jackson, 1995]이 있다. 문헌 [Thayer and Dorfman, 1999]은 요구사항분석에 대한 론문들을 종합한것이다. 서면원형의 우점은 문헌 [Rettig, 1994]에서 론의된다. 요구사항확정단계에 대한 기사들은 *Communications of the ACM* 1995년 5월호와 *IEEE Software* 1998년 3월/4월호에 있다. 요구사항분류에 비용 대 리득분석을 리용하는 문제는 문헌 [Karlsson and Ryan, 1997]에 서술되여 있다. *Communications of the ACM* 1998년 12월호에는 요구사항추적과 관련한 많은 기사들이 포함되여 있다.

신속원형의 소개와 관련한 도서들은 문헌 [Connell and Shafer, 1989; and Gane, 1989]이다. *IEEE Computer* 1995년 2월호에는 신속원형작성에 대한 많은 기사들이 포함되여 있다. 39개의 상업용제품들에서 신속원형작성의 리용과 관련한 보고 [Gordon and Bieman, 1995]는 매우 유용한 정보자원으로 된다. 문헌 [Lichter, Schneider-Hufschmidt, and Zűllighoven, 1994]은 원형작성이 중요한 역할을 수행하고 있는 5개의 산업쏘프트웨어프로젝트에 대하여 자세히 소개하고 있다. 신속원형작성모형은 신속응용개발(*rapid development*; RAD)의 한가지 판본으로 된다. RAD에 대한 여러개의 기사들이 *IEEE Software* 1995년 11월호에 제시되여 있다.

인간적인자는 문헌 [Dix, Finlay, Abowd, and Beale, 1993; Browne, 1994, and Preece, 1994]에서 론의되였다. 문헌 [Nielsen, 1993]은 사용자대면부와의 대화가 어떻게 유용성을 본질적으로 개선할수 있는가에 대하여 서술하고 있다. 문헌 [Myers and Rosson, 1992]에서는 총 쏘프트웨어개발노력의 절반이상이 제품의 사용자대면부와 관련한 부분에 돌려질수 있다는것을 지적하였다.

사용자대면부설계에 대한 기사들은 *Communications of the ACM*의 1993년 4월호에서 찾아 볼수 있다. 문헌 [Gentner and Grudin, 1996]에서는 사용자대면부설계의 기초를 이루는 모형들을 분석한다. *IEEE Software* 1997년 7,8월호에는 사용자대면부와 관련한 많은 기사들이 있는데 특히 문헌 [Sears and Lund, 1997]에서 이에 대하여 론의하고 있다. *Communications of the ACM*의 1999년 5월호에 있는 론문들의 주제는 쏘프트웨어의 리용문제이다. 콤퓨터체계에서의 인간적인자들에 대한 년차대회 회보에는 넓은 측면의 인간적인자들에 대한 중요한 정보자원들이 들어 있다.

문 제

10.1. 당신이 쏘프트웨어경영자로서 방금 Victoria & Niagara Software에 접하였다. Victoria & Niagara는 폭포모형을 성공적으로 리용하면서 소규모기업을 위한 체계프로그람을 몇년동안 개발하여 왔다. 경험에 토대하여 당신은 신속원형작성모형이 쏘프트웨어를 개발하는 아주 우월한 방법이라고 생각한다. 개발기업체가 신속원형작성에로 전환하여야 한다고 당신이 생각한 리유를 설명하는 쏘프트웨어개발팀 부총재앞으로 보내는 보고서를 작성하시오. 부총재는 한페지이상 되는 보고서를 좋아 하지 않는다는것을 기억하시오.

10.2. 당신은 Victoria & Niagara 쏘프트개발팀 부총재이다. 문제 10.1의 보고서에 대답하시오.

10.3. 당신이 리용할수 있는 프로그람작성언어들가운데서 어느것이 신속원형작성에 리용되지 말아야 하는가? 대답에 대한 리유를 말하시오.

10.4. 신속원형작성이 개발팀에 크게 도움이 되지 않을것 같은 프로젝트를 서술하시오.

10.5. 10.3에서는 신속원형작성이 사용자대면부를 개발할 때 특별히 효과적이라는것을 설명하였다. 이 점에 관하여 어떤 환경에서 의뢰자의 요구를 결정하기 위한 등가적인 다른 대책이 있는가?

10.6. 어떤 환경에서 신속원형을 세련시키는것이 리치에 맞는가?

10.7. 만일 신속원형이 빨리 구성되지 않으면 어떤 결과가 생기는가?

10.8. 만일 제품이 객체지향파라다임을 리용하여 개발되여야 한다면 신속원형을 리용하여야 하는가?

10.9. (과정안상 목표) 부록 1에 있는 브로드랜즈지역 아동병원프로젝트를 위한 신속원형을 작성하시오. 지도교원이 규정한 쏘프트웨어와 하드웨어를 리용하시오.

10.10. (실례연구) 10.4의 요구로부터 출발하여 Lisp, Smalltalk, 또는 Perl과 같은 해석형언어로 신속원형을 작성하시오.

10.11. (실례연구) 항공음식전문회사제품에 관한 신속원형이 C로 작성되였고 생산품질이 높은 실현은 C++로 작성되게 된다고 하자. C는 문법적으로 C++의 부분모임이기때문에 이 신속원형이 제품의 정식판본안으로 완성되여 들어 가는것을 막으려면 어떻게 하여야 하는가?

10.12. (실례연구) 항공음식전문회사제품에 관한 신속원형이 Java로 작성되였고 생산품질이 높은 실현도 Java로 작성되게 된다고 하자. 신속원형이 제품의 정식판본으로 완성되여 들어 가는것을 막으려면 어떻게 하여야 하는가?

10.13. (실례연구) 만일 쏘프트웨어가 당신에게 도움이 된다면 10.15의 신속원형에 도형사용자대면부를 추가하시오.

10.14. (실례연구) 10.15의 신속원형에서 기록배렬들은 승객과 비행기기록을 저장하는데 리용된다. 이것은 신속원형작성팀에 있어서 좋은 결정으로 되는가? 당신의 대답을 안받침하기 위하여 동적자료구조를 리용하여 신속원형을 기록하시오.

10.15. (실례연구) 10.15의 많은 신속원형부분프로그람들은 비여 있다. 실례로 특별식사가 한번이상 오르지 않은 매 승객들의 이름과 주소를 포함하고 있는 보고서를 인쇄하는 부분프로그람은 코드화되지 않았다. 이것은 신속원형작성팀에 있어서 좋은 결정으로 되는가? 당신은 대답을 안받침하기 위하여 비여 있는 부분프로그람의 몸체부를 완성하시오.

10.16. (쏘프트웨어공학독본) 교원이 문헌 [karlsson and Ryan, 1997]의 복제본을 배포할것이다. Karlsson과 Ryan 방법을 비용 대 리득분석에 기초하여 완성된 생명주기 모형으로 어떻게 확장할수 있는가?

참 고 문 헌

[Boehm, Gray, and Seewaldt, 1984] B. W. Boehm, T. E. Gray, and T. Seewaldt, "Prototyping versus Specifying: A Multi-Project Experiment," *IEEE Transactions on Software Engineering* **SE-10** (May 1984), pp. 290–303.

[Brooks, 1975] F. P. Brooks, Jr., *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley, Reading, MA, 1975. Twentieth Anniversary Edition, Addison-Wesley, Reading, MA, 1995.

[Browne, 1994] D. Browne, *STUDIO: STructured User-interface Design for Interaction Optimization,* Prentice Hall, Englewood Cliffs, NJ, 1994.

[Capper, Colgate, Hunter, and James, 1994] N. P. Capper, R. J. Colgate, J. C. Hunter, and M. F. James, "The Impact of Object-Oriented Technology on Software Quality: Three Case Histories," *IBM Systems Journal* **33** (No. 1, 1994), pp. 131–57.

[Connell and Shafer, 1989] J. L. Connell and L. Shafer, *Structured Rapid Prototyping: An Evolutionary Approach to Software Development,* Yourdon Press, Englewood Cliffs, NJ, 1989.

[Davis, 1993] A. M. Davis, *Software Requirements: Objects, Functions, and States,* rev. ed., Prentice Hall, Englewood Cliffs, NJ, 1993.

[Dix, Finlay, Abowd, and Beale, 1993] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human-Computer Interaction,* Prentice Hall, Englewood Cliffs, NJ, 1993.

[Gane, 1989] C. Gane, *Rapid System Development: Using Structured Techniques and Relational Technology,* Prentice Hall, Englewood Cliffs, NJ, 1989.

[Gause and Weinberg, 1990] D. Gause and G. Weinberg, *Are Your Lights On? How to Figure out What the Problem Really Is,* Dorset House, New York, 1990.

[Gentner and Grudin, 1996] D. R. Gentner and J. Grudin, "Design Models for Computer-Human Interfaces," *IEEE Computer* **29** (June 1996), pp. 28–35.

[Gordon and Bieman, 1995] V. S. Gordon and J. M. Bieman, "Rapid Prototyping Lessons Learned," *IEEE Software* **12** (January 1995), pp. 85–95.

[Jackson, 1995] M. Jackson, *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices,* Addison Wesley Longman, Reading, MA, 1995.

[Karlsson and Ryan, 1997] J. Karlsson and K. Ryan, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software* **14** (September/October 1997), pp. 67–74.

[Lichter, Schneider-Hufschmidt, and Züllighoven, 1994] H. Lichter, M. Schneider-Hufschmidt, and H. Züllighoven, "Prototyping in Industrial Software Projects—Bridging the Gap between Theory and Practice," *IEEE Transactions on Software Engineering* **20** (November 1994), pp. 825–32.

[Myers and Rosson, 1992] B. A. Myers and M. B. Rosson, "Survey on User Interface Programming," *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems,* Monterey, CA, May 1992, pp. 195–202.

[Nielsen, 1993] J. Nielsen, "Iterative User-Interface Design," *IEEE Computer* **26** (November 1993), pp. 32–41.

[Preece, 1994] J. Preece, *Human-Computer Interaction,* Addison-Wesley, Reading, MA, 1994.

[Rettig, 1994] M. Rettig, "Prototyping for Tiny Fingers," *Communications of the ACM* **37** (April 1994), pp. 21–27.

[Schach and Wood, 1986] S. R. Schach and P. T. Wood, "An Almost Path-Free Very High-Level Interactive Data Manipulation Language for a Microcomputer-Based Database System," *Software—Practice and Experience* **16** (March 1986), pp. 243–68.

[Sears and Lund, 1997] A. Sears and A. M. Lund, "Creating Effective User Interfaces," *IEEE Software* **14** (July/August 1997), pp. 21–25.

[Shneiderman, 1997] B. Shneiderman, *Designing the User Interface: Strategies for Effective Human-Computer Interaction,* 3rd ed., Addison Wesley Longman, Reading, MA, 1997.

[Thayer and Dorfman, 1999] R. H. Thayer and M. Dorfman, Editors, *Software Requirements Engineering,* rev. 2nd ed., IEEE Computer Society Press, Los Alamitos, CA, 1999.

제11장. 명세작성단계

명세서는 두개의 서로 모순되는 요구를 만족시켜야 한다. 한편 명세서는 콤퓨터전문가는 아닐수도 있는 의뢰자에게 명백하고 리해하기 쉬워야 한다. 결국은 의뢰자가 제품에 값을 지불하는데 그가 새 제품이 어떤것으로 될것인가를 실지로 리해하지 못한다면 그는 그 제품의 개발을 맡기지 않기로 결심하든가 또는 다른 쏘프트웨어개발기업체를 청하여 그것을 개발하도록 할 가능성이 크다.

다른 한편 명세서는 설계를 작성하는데 쓰이는 유일한 정보원천이기때문에 완전하고 세부적이여야 한다. 비록 의뢰자가 요구사항확정단계에서 모든 요구들이 정확하게 결정되였다고 동의하여도 만일 명세서가 생략, 모순, 애매성과 같은 오유를 포함하면 실현에로 넘어 갈 설계에서 불가피하게 오유결과가 생기게 될것이다. 그렇기때문에 의뢰자가 리해하기 쉽도록 충분히 비공학적이면서도 생명주기의 마감에 의뢰자에게 오유 없는 제품이 배포되도록 충분히 정확한 형식으로 목적하는 제품을 표현하는 기법이 필요된다. 이와 같은 명세작성기법이 바로 이 장과 다음장의 주제이다. 이 장에서는 고전적(구조화된) 명세작성기법에 중점을 두며 12장에서는 객체지향분석에 전심한다.

11. 1. 명 세 서

명세서는 의뢰자와 개발자사이의 계약서이다. 명세서에는 제품이 하여야 할것과 그 제품에 대한 제약을 명기한다. 사실 매 명세서들은 제품이 만족시켜야 할 제약을 포함하고 있다. 거의 언제나 제품을 배포하기 위한 최종기한이 명기된다. 또 하나의 공통계약은 다음과 같다. 의뢰자는 새 제품들이 실지로 명세서의 모든 측면을 만족시킨다고 할 때까지 《제품은 현존 제품과 병렬로 실행할수 있는 방식으로 설치될것이다》. 기타 제약들은 이식성을 포함할수도 있다. 즉 제품은 같은 조작체계들을 리용하는 다른 하드웨어상에서 동작하거나 각이한 조작체계들에서 동작할수 있도록 구성되여야 한다는것이다. 신뢰성이 또 하나의 제약으로 될수도 있다. 만일 해당 제품이 집중치료실에서 환자를 감시하여야 한다면 하루 24h 전적으로 동작하는것이 가장 중요하다. 응답시간이 요구로 될수 있다. 이 범주에서 전형적인 제약은 《형식 4에 대한 95%의 질문들은 0.25s이내에서 답변될것이다.》로 될수 있다. 대다수의 응답시간제약들은 응답시간이 콤퓨터의 현재 부하에 의존하기때문에 확률적인 용어로서 표현되여야 한다. 반대로 이른바 장치실시간제약들은 절대적인 술어로 표현된다. 실례로 전투기에 날아 오는 미싸일을 장치실시간의 95%에 해당한 0.25s이내에 통지하는 프로그람은 쓸모 없다. 제품은 장치실시간에 대한 100%의 제약을 만족시켜야 한다. 명세서의 중요한 요소는 합격판정기준모임이다. 의뢰자에게 제품이 명세서를 실지로 만족시키며 또 개발자의 임무가 수행된다는것을 증명하는데 리용할수 있는 일련의 시험을 자세히 해석하는것은 개발자와 의뢰자의 관점에서 모두 중요하다.

일부 합격판정기준들은 제약에 대한 재설명으로 될수 있으며 반면에 기타 판정기준

들은 여러가지 론의를 거쳐야 한다. 실례로 의뢰자는 개발자에게 제품이 조종할 자료들에 대한 서술을 제공할수 있다. 그러면 그 제품이 이 류형의 자료를 정확하게 처리하고 적합치 않은(즉 오유) 자료들을 려파해 내는것이 적당한 합격판정기준으로 될수 있다. 일단 개발팀이 문제를 완전히 리해하면 가능한 해결전략이 제안될수 있다. 해결전략(*solution strategy*)이란 제품을 만들기 위한 일반적방법이다. 실례로 어떤 제품에 대한 한가지 가능한 해결전략은 직결식자료기지를 리용하는것일수 있으며 또 다른 해결전략은 습관적인 단층파일을 리용하고 철야묶음실행을 리용하여 필요한 정보를 추출하는것일수도 있다. 해결전략을 결정할 때 명세서에 있는 제약들을 념려하지 않고 전략을 제의하는것이 좋은 방법이다. 그러면 각이한 해결전략들을 제약의 관점에서 평가할수 있고 필요한 변경을 진행할수 있다. 어떤 특정한 해결전략이 의뢰자의 제약들을 만족시킬것인가를 결정하는데는 여러가지 방법이 있다. 한가지 명백한것은 원형작성인데 이것은 3.7에서 이미 론의한바와 같이 사용자대면부들과 시간제약들과 관계된 론점들을 해결하기 위한 좋은 기법으로 될수 있다. 제약들이 만족될것인가를 결정하기 위한 기타 기법들은 모의[Banks, Carson, Nelson, Nichol, 2000]와 해석적망모형화[Kleinrock and Gail, 1996]를 포함하고 있다.

이 과정에서 많은 해결전략들이 제시되고 그다음 버려 질것이다. 버려 진 모든 전략들과 그것들이 버려 진 리유를 적은 기록을 보관하여 두는것이 중요하다. 이것은 만일 선택된 전략을 정당화하기 위하여 그 기록들을 다시 보게 되는 경우에 개발팀을 도와 주게 될것이다. 그러나 보다 중요하게는 유지정비단계에서 항시적으로 나타나는 한가지 위험이 존재한다. 즉 어떤 새롭고 현명치 못한 해결전략을 제의하려는 시도와 함께 확장과정이 생겨 나게 된다. 개발기간에 어떤 전략들이 기각되는 원인을 기록해 놓는것은 유지정비단계에서 매우 쓸모 있다.

이러한 견지로부터 개발팀은 제약들을 만족시키는 하나 또는 그이상의 가능한 풀이전략들을 결정하는것이다. 두단계결정이 만들어 져야 한다. 첫째로, 의뢰자에게 콤퓨터화를 진행할것을 권고하여야 하는가 하는것인데 만약 권고한다면 어느 해결전략을 채용하여야 하는가를 결정한다. 첫번째문제에 대한 답변은 비용 대 리득분석(5.2)에 기초하여 가장 훌륭히 결정될수 있다.

둘째로, 만일 의뢰자가 프로젝트를 실행해 나가기로 결정한다면 의뢰자는 개발팀에 의뢰자의 총적비용을 최소로 한다든가 또는 투자에 대한 리윤을 최대로 한다든가와 같은 리용될 최량화판정기준을 알려 주어야 한다. 개발자들은 그러면 의뢰자에게 어느 해결전략이 그 최량화기준을 가장 만족시키는가를 통지한다.

1 1 . 2 . 비형식적명세작성

개발프로젝트들에서 명세서는 여러페지의 영어 또는 프랑스어나 코사어와 같은 민족어로써 구성된다. 전형적인 명세서의 한 단락을 다음과 같이 제시한다.

> BV.4.2.5 만일 이달 매상고가 목표매상고보다 낮다면 보고서를 찍으며 목표매상고와 실제매상고사이 차가 전달의 목표매상고와 실제매상고사이 차의 절반보다 작지 않다면 또는 목표

매상고와 이달의 실제매상고사이 차가 5%이하라면 보고서를 찍는다.

이 단락의 배경은 다음과 같다. 련쇄된 소매상점들에 대한 관리자측은 매 상점에 관한 매달 목표매상고를 설정하며 만일 어떤 상점이 이 목표를 만족시키지 못하면 보고서가 인쇄되게 된다. 다음과 같은 대본을 고찰하자. 가령 1월달 어떤 상점의 판매목표가 10만딸라인데 실제 매상고는 6만 4천딸라이라고 하자. 즉 목표보다 36% 낮다. 이 경우에 보고서가 인쇄되여야 한다. 이제 2월달 목표매상고는 12만딸라이며 실제매상고는 겨우 10만딸라이라고 가정하자. 즉 목표보다 16.7% 낮다. 비록 판매액이 목표보다 낮지만 2월달 퍼센트차 16.7%는 전달의 36%의 절반보다 적으며 따라서 관리자측은 일정한 개선이 있다고 믿고 보고서를 인쇄하지 않는다. 다음으로 3월달에 목표는 10만딸라이지만 상점에서 판매액이 9만 8천딸라로서 목표보다 다만 2% 낮다고 가정하자. 퍼센트차가 5%보다 적으므로 보고서는 인쇄되지 말아야 한다.

고찰하고 있는 명세서단락을 주의 깊게 다시 읽으면 련쇄된 소매상점관리자측이 실지 요구하는것으로부터 좀 탈선하였다는것을 알수 있다. BV. 4.2.5단락에서는 목표매상고와 실제매상고사이차에 대하여 말하고 퍼센트차는 언급하지 않고 있다. 1월달에 그 차는 36,000딸라 2월달에는 200,000딸라이다. 관리자측이 요구하는 퍼센트차는 1월달에 36%부터 2월달에 16.7% 떨어 졌다. 그러나 실제차는 36,000로부터 2,000딸라로 떨어 졌으며 이것은 36,000딸라의 절반보다 더 크다. 그러므로 만일 개발팀이 요구명세서를 충실히 실현하였다면 보고서를 인쇄해야 하였는데 이것은 관리자측이 바라는것이 아니다.

그다음 마지막절에서는 《…[의]차가 5%》라고 말하고 있는데 이것은 물론 5%의 퍼센트차를 의미하고 있으며 다만 단어 퍼센트가 요구명세서단락의 어디에서도 나타나지 않은것이다.

그렇기때문에 명세서에는 많은 오유를 포함하고 있다.

첫째로, 의뢰자가 원하는것이 무시되였으며 둘째로, 애매성이 존재한다. 즉 마지막절은 《…[의] 퍼센트차가 5%》든지 《…의 차는 5,000딸라》이라고 읽어야 한다. 그밖에 형식이 불충분하다. 이 단락이 말하고 있는것은 《만일 무슨 현상이 일어 나면 보고서를 인쇄하라. 그러나 만일 그밖에 무슨 현상이 일어 나면 그것을 인쇄하지 말라. 그리고 만일 세번째 현상이 일어 나면 보고서를 인쇄하지 말라.》이다. 만일 이 명세서가 보고서가 언제 인쇄되야 하는가를 간단히 서술하였더라면 훨씬 명백하였을것이다. 결국 단락 BV.4.2.5는 명세서를 어떻게 작성하여야 하는가에 대한 그리 좋은 실례로는 되지 않는다.

사실 BV.4.2.5는 허구적이지만 공교롭게도 아주 많은 요구서들에서 전형적인것이다. 이 실례가 부적당하며 이러한 부류의 문제는 명세서를 전문명세작성자가 주의 깊게 작성한다면 생기지 않을수 있다고 생각할수 있다. 이러한 론의를 반박하기 위하여 제6장의 실례연구를 여기서 다시 시작하기로 한다.

11．2．1．실례연구 : 본문처리

6.5.2로부터 1969년에 나우르가 정확성증명에 관한 론문을 썼다는것을 상기하자[Naur,

1969]. 그는 자기의 기법을 본문처리문제로써 례증하였다. 이 기법을 리용하여 나우르는 이 문제를 해결하기 위한 ALGOL60절차를 작성하였으며 이 절차의 정확성을 비형식적으로 증명하였다. 나우르(Naur)의 론문에 대한 조사자들은 이 절차에서 한가지 오유를 지적하였다[Leavenworth, 1970]. 론돈은 그이후 나우르의 절차에서 3개의 보충적인 오유를 발견하고 정확한 절차를 제시하고 그것의 정확성을 형식적으로 증명하였다[London, 1971]. 구데나우(Goodenough)와 게르하르트(Gerhart)는 그이후 론돈(London)이 찾지 못한 3개의 오유를 더 찾아 냈다[Goodenough and Gerhart, 1975]. 론돈, 구데나우와 게르하르트와 같은 조사자들이 찾아 낸 총 7개의 오유가운데서 두개는 명세작성오유로서 고찰할수 있다. 실례로 나우르의 명세서는 입력이 두개의 련이은 중단점(공백 또는 개행문자)들을 포함하면 무슨 현상이 일어 나는가를 설명하지 않고 있다. 이 리유로 하여 구데나우와 게르아르트는 새로운 명세서모임을 만들어 냈다. 그들의 명세서는 6.5.2에 제시된 나우르의 명세서보다 약 4배 더 길다.

1985년이 메이어(Meyer)는 형식적명세작성기법에 대한 론문을 썼다[Meyer, 1985]. 이 론문에 대한 중요한 혹평은 명세서가 영어와 같은 자연언어로 작성되여 모순, 애매성, 생략을 가질수 있다는것이다. 그는 명세서를 형식적으로 표현하기 위하여 수학적용어를 리용할것을 권고하였다. 메이어는 구데나우와 게르하르트의 명세서에서 12개의 오유를 찾고 이 문제를 수정하기 위한 수학적명세서모임을 개발하였다. 그다음 메이어는 자기의 수학적명세서를 의역하여 영어로 된 명세서로 작성하였다. 우리의 견해에 의하면 메이어의 영어명세서는 한가지 오유를 포함하고 있다. 메이어는 자기의 론문에서 만일 행당 최대문자수가 10이고 실례로 입력문장이 WHO WHAT WHEN이면 나우르 및 구데나우와 게르하르트의 명세서에 관하여 두개의 등가적으로 타당한 출력이 존재한다는것을 지적하였다. 즉 첫 행에서의 WHO WHAT와 두번째 행에서의 WHEN 또는 첫행에서의 WHO와 두번째 행에서의 WHAT WHEN은 등가이다. 사실 메이어가 의역한 영어명세서도 이러한 애매성을 포함하고 있다.

중요한것은 구데나우와 게르하르트의 명세서가 최대의 심중성을 가지고 작성되였다는것이다. 결국 그것들은 나우르의 명세서를 수정하기 위하여 작성되였다. 더우기 구데나우와 게르하르트의 론문은 두개의 판본으로 출판되였는데 첫번째는 련관된 대회회보에 출판되였고 두번째는 련관된 잡지에 출판되였다[Goodenough and Gerhart, 1975]. 결국 구데나우와 게르하르트는 둘 다 일반적으로는 쏘프트웨어공학의 전문가들이고 개별적으로는 명세작성자들이다. 그러므로 만일 두명의 전문가들이 자기들이 요구한 시간을 가지고 메이어가 12개의 오유를 발견한 명세서를 주의 깊게 만들었다면 주어 진 시간하에서 작업하는 일반콤퓨터전문가는 오유 없는 명세서를 작성하는데 얼마만한 기회를 가질수 있는가? 보다 나쁘게는 본문처리문제는 25 또는 30행으로 코드화될수 있지만 실세계제품은 수만 지어는 수백만행의 원천코드로 구성될수 있다.

명백히 자연언어는 제품을 평가하기 위한 좋은 방법으로 되지 않는다. 이 장에서는 보다 좋은 대응방법이 고찰된다. 명세작성기법을 비형식적인것으로부터 보다 형식적인것을 서술하는 형식으로 설명한다.

1 1. 3. 구조화체계분석

쏘프트웨어를 명기하기 위하여 도형을 리용한것은 1970년대의 중요한 기법이였다. 도형를 리용하는 3개의 기법이 특히 보편적이였는데 그것들의 데마르코[DeMarco, 1978], 게인과 사르센[Gane and Sarsen, 1979], 요르돈과 콘스탄틴[Yourdon and Constantin, 1979]의 기법이다. 이 세가지 기법은 모두 동등하게 좋으며 본질적으로 등가이다. 게인과 사르센의 표시법이 산업에서 현재 가장 광범히 리용되고 있기때문에 그들의 방법을 제시한다.

이 방법을 리해하기 위하여 다음의 실례를 고찰하자.

1 1. 3. 1. 쏘프트웨어상점

쌜리(Salley)의 쏘프트웨어상점은 여러 공급자들로부터 쏘프트웨어들을 사들이여 판매한다. 쌜리는 대중적인 쏘프트웨어패키지들을 사들이며 필요할 때 다른 제품들을 주문한다. 쌜리는 보험회사, 주식회사, 일부 개별적사람들에게 신용대부를 확장한다. 쌜리의 쏘프트웨어상점은 경기가 좋은데 개당 250딸라의 평균소매가격으로서 매달 300패키지의 쏘프트웨어를 넘긴다. 기업운영에서 성공하고 있음에도 불구하고 쌜리는 콤퓨터화를 실현하라는 권고를 받았다. 쌜리가 콤퓨터화를 하여야 하는가?

서술한바와 같이 이 문제는 적당치 않다. 다음과 같은것을 해석하여야 한다. 다음의 업무기능 즉 지출능력계산, 수용능력계산, 제품목록 등에서 어느 업무기능들을 콤퓨터화해야 하는가? 비록 그것이 충분치 않다고 하더라도 체계는 계렬처리식으로 되여야 하는가 직결식으로 되여야 하는가?

국내콤퓨터체계로 되여야 하는가 아니면 해외조달로 리용되여야 하는가? 비록 문제가 더 세련된다고 할지라도 여전히 기본론점은 놓치고 있다. 즉 업무를 콤퓨터화할 때 쌜리의 목적이 무엇인가 하는것이다.

쌜리의 목적이 알려 질 때에만 분석을 계속해 나갈수 있다. 실례로 그 녀자가 쏘프트웨어를 팔기때문에 단순히 콤퓨터화를 하려고 한다면 그는 콤퓨터의 능력들을 화려하게 돋보이게 하는 여러가지 음향 및 빛효과를 가진 실내콤퓨터체계를 요구한다. 다른 한편 만일 쌜리가 《부당한》돈을 세척할 목적으로 자기 업무를 리용하려 한다면 네댓개의 서로 다른 장부책을 만들어 놓고 아무런 결산결과도 남기지 않는 쏘프트웨어제품을 요구한다.

이 실례에서는 쌜리가 《더 많은 돈을 벌기 위하여》 콤퓨터화를 진행하려고 한다고 가정하고 있다. 이것이 그리 도움이 되지 않지만 비용 대 리득분석이 이 세개의 업무항목모두(또는 어느 하나)를 콤퓨터화하겠는가를 결정할수 있다는것은 명백하다.

여러가지 표준적인 방법들에서 제기되는 중요한 위험성은 사람들이 이를테면 10GB의 LimeⅢ콤퓨터와 레이자인쇄기와 같은 해결방안을 먼저 제기하고 문제는 후에 찾아내려고 한다는것이다. 반대로 게인과 사르센은 9계단 기술을 리용하여 의뢰자의 요구를 분석한다[Gane and Sarsen , 1979]. 한가지 중요한 점은 계단식세련(개량)이 이 9계단의 대부분에서 리용된다는것이다. 이것은 이 기법이 현시될 때 지적될것이다.

쌜리의 요구를 결정할 때 구조화분석의 첫 단계는 물리적자료흐름에 대립하는것으로

서 론리적자료흐름(*logical data flow*)을 결정하는것이다(즉 그것이 어떻게 발생했는가에 대립하는것으로서 무엇이 발생했는가 하는것이다.). 이것은 자료흐름도(DFD)를 작성함으로써 수행된다. DFD는 그림 11-1에 보여 준 4개의 기본기호를 리용한다(게인과 사르센의 표시법은 데마르코[DeMarco, 1978]과 요르돈과 콘스탄틴[Yourdon and Constantine, 1979]의 표시법과 류사하지만 리상적이 아니다.).

1단계. DFD를 작성한다 임의의 단순하지 않은 제품에 대한 DFD는 규모가 클수 있다. DFD는 론리적자료흐름의 모든 측면들에 대한 하나의 그림표현이며 그것은 또 7±2보다 상당히 많은 요소들을 포함한다고 약속되여 있다. 이러한 리유로 하여 DFD는 계단식세련방법(5.1)으로 개발되여야 한다.

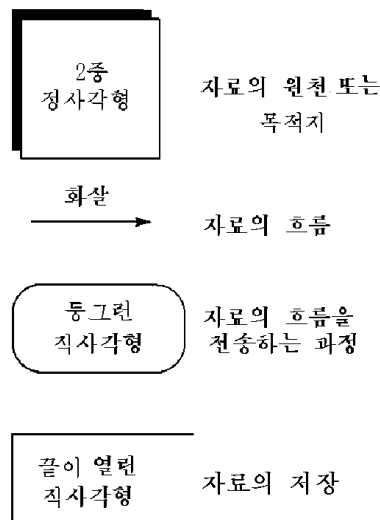



그림 11-1. 게인과 사르센의 구조화체계분석의 기호

자료흐름도는 요구문서나 신속원형내에서 자료의 흐름을 인식함으로써 구성된다. 자료의 매 개별적흐름은 자료원천-목적지(2중정사각형통으로 표시된다.)든가 자료의 저장(끝이 열린 직사각형으로 표시된다.)으로 시작되고 끝난다. 자료는 하나 또는 그이상의 공정(둥그런 직사각형으로 표시)에 의하여 변환된다. 매 련속적인 세 련에서 자료의 새로운 흐름이 DFD에 추가되든가 보다 세부적인 사항을 추가하는것으로써 자료의 현존흐름이 개선된다.

실례를 고찰하자. 첫번째 개선을 그림 11-2에 보여 주었다. 이 론리적자료흐름은 여러가지로 해석할수 있다. 두가지 가능한 실현은 다음과 같다.

실현 1에서 자료저장패키지자료는 책상서랍안에 있는 많은 사용설명서들과 사전에 전시된 디스크나 CD판들이 들어 있는 약 900개의 수축포장한 통들로 구성된다. 자료저장 **손님자료**는 고무띠로 묶은 5×7"카드들의 모임이며 거기에 지불기간이 넘은 손님들에 대한 목록이 더해 졌다. 과정(작용) 주문처리는 쎌리가 시렁우에 놓인 적당한 패키지를 찾는것인데 만일 그것을 사용설명서에서 찾는것이 필요하다면 정확한 5×7"카드를 찾고 손

님의 이름이 채납자명단에 없는가를 검사한다. 이 실현은 완전히 수동적이며 쌜리가 현재 자기 업무를 진행하고 있는 방법에 대응한다.

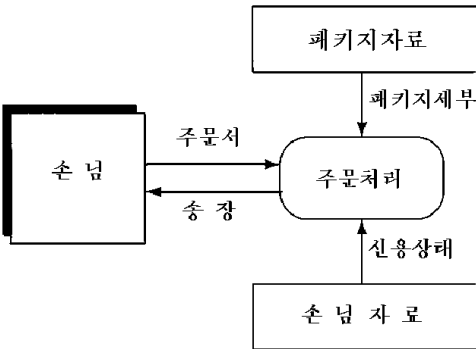



그림 11-2. 자료흐름도: 첫번째 세련

실현 2에서 자료저장패키지자료와 **손님자료**는 콤퓨터파일이며 주문처리는 쌜리가 말단콤퓨터에서 손님의 이름과 패키지이름을 입력하는것이다. 이 실현은 모든 정보를 직렬로 리용할수 있는 완전히 콤퓨터화된 해결방안에 대응한다.

그림 11-2의 DFD는 앞에서 설명한 두가지 실현뿐아니라 다른 무한한 가능성을 표현하고 있다. 관건적인 점은 DFD가 정보의 흐름을 표현하고 있다는것이다. 즉 쌜리의 손님이 원하는 실제적인 패키지는 흐름에서 중요치 않다.

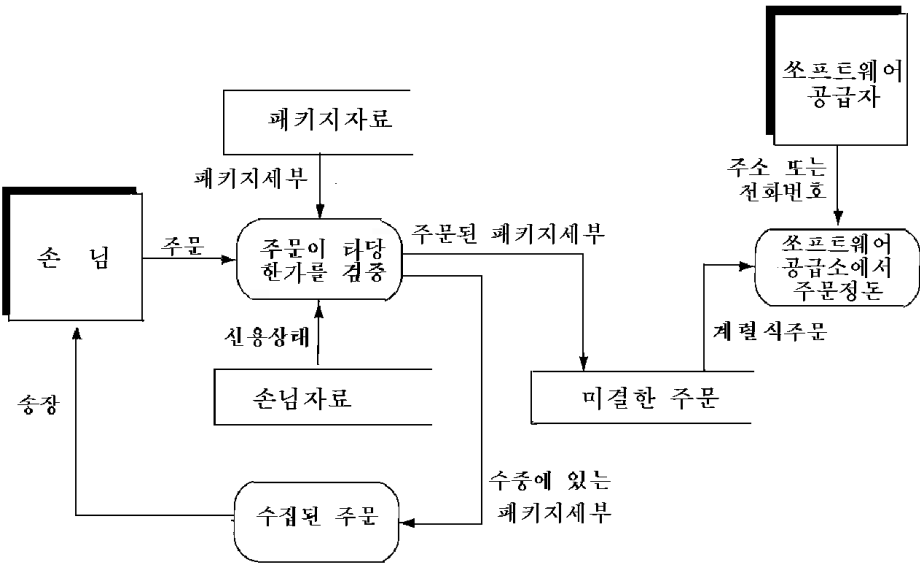



그림 11-3. 자료흐름도: 두번째 세련

DFD는 이제 계단식으로 세련된다. 두번째 세련은 그림 11-3에 묘사된다. 손님이 쌜

리의 손에 들지 않은 패키지를 요구할 때 무슨 현상이 일어 나는가를 표시하는 자료의 론리적흐름이 DFD에 추가된다. 특히 그 패키지에 대한 세부자료는 자료저장 **미결한 주문**에 위치하고 있는데 이것은 하나의 콤퓨터파일일수 있지만 이 단계에서는 충분히 마닐라 폴더(*manila folder*)일수 있다. 자료저장 **미결한 주문**은 콤퓨터 또는 쌜리가 매일 조사하며 만일 한명의 공급자에게 충분한 주문이 있다면 하나의 계렬처리주문이 놓이게 된다. 또한 만일 어떤 주문이 5일동안 대기되였다면 아무리 많은 패키지들이 련관된 공급자들로부터 주문되기를 기다리고 있다 하여도 그것이 주문된다. 이 DFD는 쏘프트웨어패키지들이 언제 공급자로부터 출발하는가 하는 론리적자료흐름을 보여 주지 않으며 또 지불능력 및 수용능력과 같은 기능들을 보여 주지 않는다. 이것들은 세번째 세련단계에서 추가된다.

DFD는 규모가 크기때문에 그림 11-4에서는 세번째 단계의 일부분만을 제시한다. 이 세련단계에서 수용능력과 관계되는 자료의 론리적흐름이 DFD에 추가된다. DFD의 나머지 부분은 지불능력과 쏘프트웨어공급자들에게 관계되여 있다.

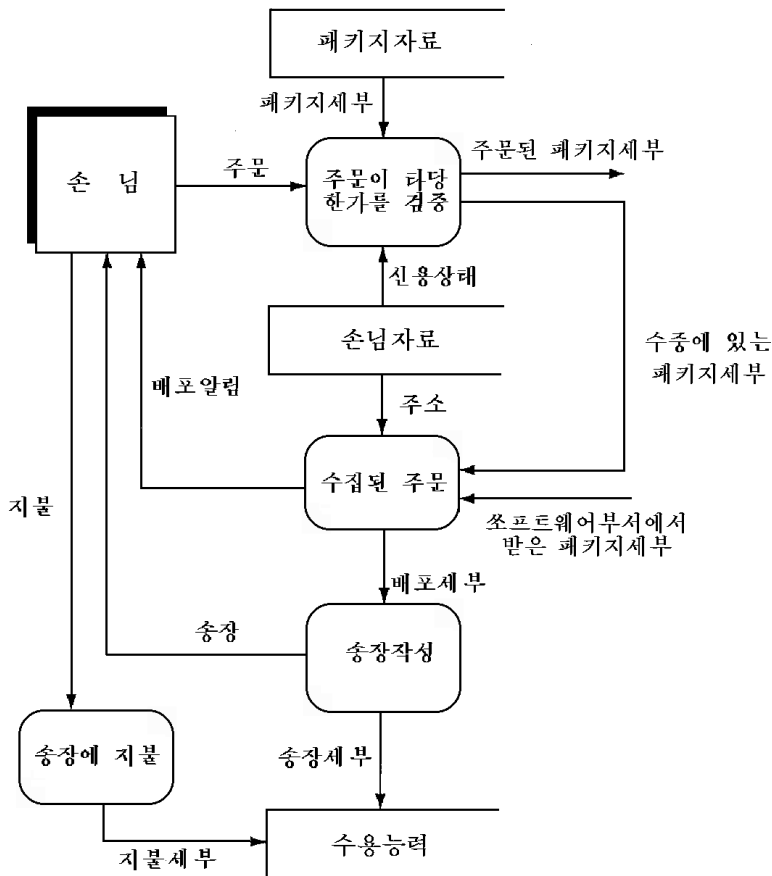



그림 11-4. 자료흐름도: 세번째 세련의 부분

최종적인 DFD는 여전히 크며 대략 6페지에 달한다. 그러나 쌜리에게 그것은 쉽게 리해될것이다. 쌜리는 그것이 자기 업무에서의 자료의 론리적흐름을 정확히 표현하고 있다고 확인하면서 거기에 수표를 한다(11.14은 DFD를 구성하는 방법에 대한 서술을 포함한다.).

제품이 크면 클수록 DFD는 더 커지게 된다.

이 절에서 우리는 쌜리의 쏘프트웨어상점을 위한 DFD의 구성을 제시하였다. 자료흐름도의 구성에 대한 보다 자세한 실례는 11.14에 서술한다.

2단계. 어느 부분을 어떻게 (계렬처리식 또는 직결식으로) 콤퓨터화하겠는가를 결정한다
무엇을 자동화하겠는가 하는 선택은 흔히 얼마나 많은 의뢰자에게 보내기 위하여 준비하여야 하는가에 의존한다. 명백히 전체 조작을 자동화하는것이 좋지만 이에 드는 비용이 너무 많다. 어느 부분을 자동화하겠는가를 결정하기 위하여 매 부분을 콤퓨터화하기 위한 여러가지 가능한 단계들에 비용 대 리득분석이 적용된다. 실례로 DFD의 매 부분에 대하여 조작들의 묶음이 계렬처리식으로 수행되여야 하는가 직결식으로 수행되여야 하는가를 결정하여야 한다. 처리할 량이 크고 안정한 조종이 요구될 때에는 흔히 계렬식처리가 좋지만 처리할 량이 작고 실내콤퓨터체계라면 직결식처리가 더 좋을수 있다. 고찰하는 실례에서 한가지 대책안은 지불능력을 계렬처리식으로 자동화하고 주문을 직결식으로 비준하는것이다. 두번째 대책안은 주문을 직결 또는 계렬식으로 처리하는것에 대처하여 쏘프트웨어공급자의 위탁판매문서들의 편집을 비롯하여 모든것을 자동화하고 나머지조작들을 직결식으로 처리하는것이다. 중요한 점은 DFD가 앞에서 설명한 모든 가능성에 대응한다는것이다. 이것은 설계단계까지 기다리지 않고 명세작성단계에서 문제를 해결하기 위한 방법에 관하여 전념하지 않는것과 일치한다.

게인과 사르센의 기법의 다음 세단계는 자료의 흐름(화살표), 처리(둥근사각형), 자료저장(열린 사각형)의 계단식세련이다.

3단계. 자료흐름의 세부를 결정한다 우선 무슨 자료항목들을 각이한 자료흐름안에 넣어야 하는가를 결정한다. 그다음 매개의 흐름을 계단식으로 세련시킨다.

실례에서 자료흐름 order는 다음과 같이 세련될수 있다.

order:
  order identification
  customer details
  package details

다음으로 우에 서술한 order의 매 요소들을 더욱 세련시킨다. 규모가 큰 제품인 경우에 자료사전(5.4)에 모든 자료요소들의 통로를 보관해 둘수 있다. 그림 11-5는 쌜리의 쏘프트웨어상점을 콤퓨터화하는데서 필요한 자료요소들에 대한 전형적인 정보를 보여 주고 있는데 그것은 자료사전안에 저장되게 된다.

4단계. 처리의 론리를 결정한다 제품안의 자료요소들이 결정되였다고 하면 다음은 매 처리공정에서 무엇이 발생하는가를 연구해야 한다. 실례로 하나의 처리공정 **교육단체에 할인을 준다**를 고찰하자. 쌜리는 쏘프트웨어개발자들에게 자기가 교육단체에 줄 할인액에

대한 세부를 제공하여야 한다. 실례로 4개까지의 패키지에 대하여서는 10%, 5개이상에 대하여서는 15% 등으로 준다. 자연언어로 된 명세서의 난점에 대처하기 위하여 그것을 문장으로부터 결정나무로 변환하여야 한다. 이와 같은 결정나무를 그림 11-6에 보여 주었다.

자료요소의 이름	식　별	해　설
order	포함하는 항목: order_identification customer_details customer_name customer_address … package_details package_name package_address …	이 항목들은 주문의 모든 세부를 포함한다
order_identification	12자리옹근수	절차 generate_order_number에 의하여 생성되는 유일한 수 첫 10개 수자는 주문개수를 나타내고 마지막 2개 수자는 검사수자이다.
verify_order_is_valid	처리절차: 입력파라메터: order 출력파라메터: number_of_errors	이 절차는 입력이 order이고 매 항목의 타당성을 검사한다. 발견된 매 오유에 대하여 적당한 통보문이 화면에 현시된다(발견된 전체 오유수는 파라메터 number_of_errors 를 되돌린다.).

그림 11-5. 쌜리의 쏘프트웨어상점을 위한 대표적인 자료사전입력점

결정나무는 모든 가능성들이 고려되였다는것을 쉽게 검사할수 있게 하는데 특히 복잡한 경우에는 더욱 그렇다. 한가지 실례를 그림 11-7에 보여 주었다. 그림 11-7로부터 꼴문구역에 있는 좌석에 앉는 학생에 대한 가격이 명시되지 않았다는것이 인차 명백하여진다. 처리공정을 표현하기 위한 또 한가지 좋은 방법은 결정표를 작성하는것이다[Pollack, Hicks and Harrison, 1971]. 결정표는 일부 CASE도구들이 결정표의 내용들을 자동적으로 콤퓨터에 입력되게 하므로 제품에서 그 부분에 대하여서는 코드로 작성할 필요가 없다는 점에서 결정나무보다 우월성을 가진다.

5단계. 자료저장을 정의한다 이 단계에서는 매 저장의 정확한 내용과 그것의 표현방법(형식)을 정의하는것이 필요하다. 결국 만일 제품이 COBOL로 실현되게 된다면 이 정보는 **pic**준위아래에서 주어 져야 한다. 만일 Ada가 리용된다면 **digits** 또는 **delta**로 명시되여야 한다. 게다가 즉시접근이 어디에서 요구되는가를 명시하는것이 필요하다.

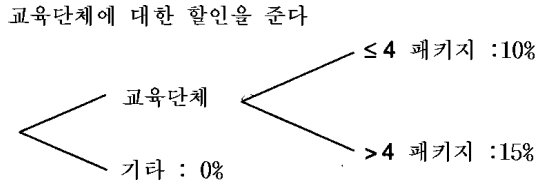



그림 11-6. 쌜리의 쏘프트웨어상점의 교육단체에 대한 할인을 나타내는 결정나무

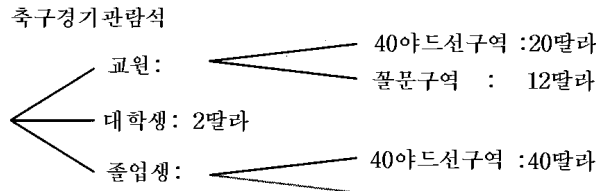



그림 11-7. 단과대학축구경기관람석값을 나타내는 결정나무

즉시접근(*immediate access*)에 대한 론의는 무슨 질문들이 제품에 첨부되게 되여 있는가에 관련되여 있다. 례하면 실례에서는 주문을 직결식으로 비준한다고 결정되여 있다고 가정하자.

손님은 이름(《당신은 Lotus 1-2-3을 재고품으로 가지고 있는가.》), 기능(《무슨 회계프로그람묶음을 가지고 있는가?》), 기계(《당신은 786을 위한 무슨 새로운 제품을 가지고 있는가?》)로 프로그람묶음을 주문할수 있지만 가격(《무슨 149.50딸라짜리제품을 가지고 있는가?》)으로는 거의 주문하지 않는다. 그렇기때문에 패키지자료에 대한 즉시접근은 이름, 기능, 기계를 요구한다.

이것은 그림 11-8의 자료즉시접근도(DIAD)에 묘사된다.

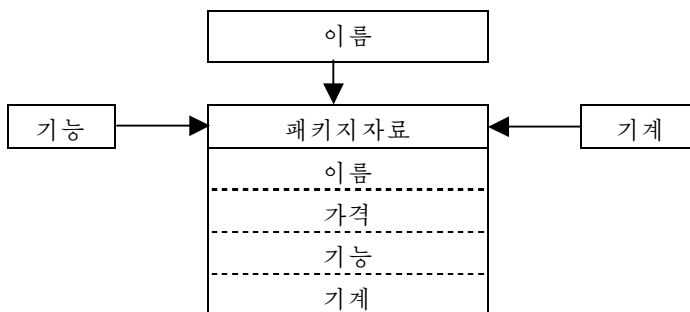



그림 11-8. 패키지자료를 위한 자료즉시접근도(DIAD)

6단계. 물리적자원을 정의한다 이제 개발자가 무엇이 직결식으로 요구되는가와 매 요소들의 표현(형식)을 알고 있다고 하면 블로크화인수(*blocking factor*)를 고려하여 한가지 결정을 할수 있다. 더우기 매 파일에 대하여 다음의 사항을 명시할수 있다. 즉 파일이

름, 조직, 기억매체, 마당준위아래의 기록들을 명시한다. 만일 자료기지관리체계(DBMS)가 리용된다면 매 표에 대한 관계정보는 이 단계에서 명시된다.

7단계. 입력출력명세서를 결정한다 만일 명세서륜곽이 자세하지 않다면 적어도 요소들에 관하여 입력형식들을 명시하여야 한다. 입력화면들이 류사하게 결정되여야 한다. 인쇄되는 출력도 가능한한 상세하게 명시되여야 하며 한편 길이가 평가되여야 한다.

8단계. 규격을 결정한다 9단계에서 장치적요구들을 결정하는데 리용될 수값자료를 계산하는것이 필요하다. 이 자료들을 입력의 크기(날자적으로 또는 시간적으로), 매 인쇄보고서의 빈도와 기한, CPU와 대용량기억장치사이를 통과하게 될 매 류형의 레코드크기와 개수, 매 파일의 크기를 포함하고 있다.

9단계. 장치적요구들을 결정한다 8단계에서 결정된 디스크파일에 있는 규격정보로부터 대용량기억장치에 대한 요구를 계산할수 있다. 게다가 여벌복사(*backup*)를 위한 대용량기억장치요구들을 결정할수 있다. 입력크기에 대한 지식으로부터 이 분야에서의 요구가 발견될수 있다. 인쇄보고서의 행수와 빈도가 알려 지기때문에 출력장치들이 명시될수 있다. 만일 의뢰자가 이미 장치들을 가지고 있다면 이 장치들이 적당할것인가 또는 추가적인 장치들을 구입해야 하는가가 결정될수 있다. 다른 한편 만일 의뢰자에게 적합한 장치가 부족하다면 무엇을 얻어야 하며 그것을 구입해야 하는가 빌려 써야 하는가에 대한 권고를 명시할수 있다.

장치적요구를 결정하는것은 게인과 사르센의 명세작성기법에서 최종단계이며 이 실례를 아래의 란에 개괄하여 준다. 의뢰자가 찬성한다면 결과적인 명세서를 설계하여 넘겨 주며 쏘프트웨어개발공정을 계속 수행한다. 많은 장점이 있음에도 불구하고 게인과 사르센의 기법은 모든 문제들에 답변을 주지는 못한다. 실례로 이 기법은 응답시간을 결정하는데 리용될수 없다. 입력-출력통로의 수는 기껏해야 대략적으로 평가될수 있다. 또한 CPU의 규격과 박자는 임의의 정확도로 평가될수 없다. 이것들이 게인과 사르센의 기법에 고유한 약점인데 공정하게 말하면 이것은 사실상 다른 모든 명세작성이나 설계의 약점이기도 하다. 그러나 명세작성단계의 끝에서 정확한 정보를 리용할수 있든지 없든지간에 장치적결정을 하여야 한다. 이와 관련한 정황은 과거에 진행한것보다는 상당히 락관적이다. 명세작성을 위한 방법론적기법이 제기되기전에 쏘프트개발공정의 시작단계에서 장치를 고려한 결정들이 옳게 채택되였다. 게인과 사르센의 기법은 제품을 명시하는 방법에서 중요한 개선을 가져 왔으며 게인과 사르센 그리고 그 기법과 경쟁하고 있는 대다수 저자들이 시간을 변수로서 본질적으로 무시하고 있다는 사실은 쏘프트웨어산업이 가져 온 우점들을 약화시키지 않고 있다.

구조화체계분석방법

- 자료흐름도를 작성한다.
- 무슨 부분을 어떻게(계렬처리식 또는 직결식) 콤퓨터화하는가를 결정한다.
- 자료흐름의 세부를 결정한다.
- 처리의 론리를 정의한다.
- 자료저장을 정의한다.
- 물리적자원을 정의한다.
- 입력-출력명세서를 결정한다.
- 규격화를 진행한다.
- 장치적요구를 결정한다.

11. 4. 기타 준형식적기법

게인과 사르센의 기법은 명백히 명세서를 자연언어로 작성하는것보다는 더 형식적이다. 동시에 그것은 다음의 론의에서 제시하는 페트리망(11.7)과 Z(11.8)와 같은 많은 기법들보다는 덜 형식적이다. 다트와 그의 동업자들은 명세서 및 설계작성기법을 비형식적, 준형식적, 형식적인것으로 분류하고 있다[Dart, Ellison, Feiler and Habermann, 1987]. 이 분류에 의하여 게인과 사르센의 구조화체계분석은 준형식적기법으로 되며 반면에 이 단락에서 언급하는 다른 두가지 기법들은 형식적기법들로 된다.

구조화체계분석이 광범히 리용되고 있다. 구조화체계분석이나 그것의 변종방법들을 리용할수 있는 좋은 기회가 있다. 그러나 다른 훌륭한 준형식적기법들도 많다. 실례로 쏘프트웨어명세작성과 설계에 대한 각이한 국제연구토론회회보들을 보시오. 서술공간의 제약으로 인하여 여기에서 제시하는 모든 방법들은 몇가지 잘 알려 진 기법들에 대한 간단한 서술에 지나지 않는다.

PSL/PSA[Teichroew and Heshey, 1977]은 정보처리제품을 명시하기 위한 콤퓨터지원기법이다. 이 이름은 이 기법의 두가지 구성요소로부터 유래되였다. 즉 제품을 서술하는데 리용된 문제서술언어(*problem statement language*; PSL)와 PSL서술을 자료기지에 입력하고 요구에 대한 보고서를 생성하는 문제서술분석기(*problem statement analyzer*; PSA)로부터 유래되였다. PSL/PSA는 광범히 리용되고 있는데 특히 제품을 문서화하는데서 많이 리용된다.

SADT[Ross, 1985]는 두개의 호상관련된 요소들로 구성되여 있다. 즉 구조분석(*structural analysis*; SA)이라고 명명한 통 및 화살표(*box-and-arrow*)도식언어와 설계기법(*design technique*; DT)로부터 SADT로 명명되였다. 게인과 사르센의 기법을 보다 크게 확장하는데서 계단식세련방법이 SADT를 밑받침하고 있다. 밀러의 법칙을 준수하는데 의식적인 노력을 기울였다. 로스가 제시한바와 같이 《무엇인가 말할만한 가치가 있는것에 대하여 말할만한 가치가 있는것은 6개 또는 그보다 적은 단락으로 표현되여야 한다.》[Ross, 1985]. SADT는 광범한 종류의 제품들 특히 복잡하고 규모가 큰 제품들을 명시하는데 성과적으로 리용되였다. 다른 많은 류사한 준형식적기법들과 마찬가지로 SADT의 실시간체계에 대한 응용성은 그리 명확치 않다.

다른 한편 SREM(*software requirements engineering method*; "shrem"이라고 발음한다.)은 명백히 어떤 작용이 출현하지 않는 조건들을 명시하기 위하여 설계되였다[Alford, 1985]. 이 리유로 하여 SREM은 실시간체계를 명시하는데 특별히 쓸모 있었으며 분산체계들에로 확장되였다. SREM은 많은 요소들로 이루어 진다. RSL은 하나의 명세작성언어이다. REVS는 명세작성과 관련된 여러가지 과제들을 수행하는 도구들의 모임인데 이러한 과제들을 RSL명세서로 어떤 자동화된 자료기지로 변환하는것, 자료흐름의 일치성을 자동적으로 검사하는것(거기에 어떤 값이 할당되기전에 그 어떤 자료항목도 리용되지 않는다는것을 확인하는것), 명세서들이 정확성을 확인하는데 리용될수 있는 모의기를 명세서로부터 발생하는것 등이다. 이밖에 SREM은 하나의 설계기법 즉 분산콤퓨터설계체계 DCDS(*distributed computing design system*)를 가지고 있다.

SREM의 능력은 전체 기법들의 기초를 이루고 있는 모형에서 흘러 나오는데 그것이

바로 11.6에서 설명하는 유한상태기계 FSM(*finite state machine*)이다. SREM의 초석으로 되고 있는 이 형식적모형의 결과로써 앞에서 언급한 일관성검사를 진행하며 개별적요소들의 성능이 주어 졌을 때 제품에 대한 성능제약이 총체적으로 만족될수 있다는것을 검증하는것이 가능할수 있다. SREM은 미공군에서 두가지 C^4I(*command, control, communication, and intelligence*)체계들을 명시하는데 리용되였다[Scheffer, Stone, and Rzepka, 1985]. SREM이 명세작성단계에서 많이 리용된다고 증명되였다고 할지라도 생명주기의 뒤단계에서 채용되는 REVS가 덜 쓸모 있다고 고찰된것 같다.

11. 5. 실체-관련모형화

구조화체계분석에서 중점은 만들어 질 제품의 자료가 아니라 그 작용들이다. 명백히 제품의 자료도 모형화되지만 자료는 작용에 비하여 2차적이다. 이와는 대비적으로 실체-관련모형화(*entity-relationship modeling*; ERM)는 제품을 명시하기 위한 준형식적인 자료지향기법이다. 제품의 응용령역에서 중점은 자료에 있다. 물론 작용들이 자료에 접근하는것이 필요하며 자료기지는 접근시간을 최소로 하는것과 같은 방식으로 조직되여야 한다. 그러나 자료에 대하여 수행된 작용들은 그리 중요하지 않다.

실체-관련모형화는 새로운 기술이 아니다. 즉 그것은 1976년이래 광범히 리용되여 왔다[Chen, 1976]. 현재 실체-관련모형화는 새롭게 부활되고 있다. 왜냐하면 그것이 제12장에서 자세히 서술되는 객체지향분석의 한가지 요소로 되기때문이다.

한가지 실체-관련도가 그림 11-9에 제시되는데 그것은 저자, 자서전, 독자들사이의 관계를 모형화하고 있다. 여기에는 세개의 실체: **저자**, **자서전**, **독자**가 있다. 제일 웃관계 **쓰기**는 저자가 자서전을 읽는다는것을 반영하고 있다. 이것은 1대 n(*one-to-many*)관계이다. 왜냐하면 한명의 저자가 하나이상의 자서전을 읽을수 있기때문이다. 이것은 저자다음의 1과 자서전 다음의 n으로서 반영된다. 실체-관련도에서는 또한 자서전과 독자사이의 두개의 관계를 보여 주고 있다.

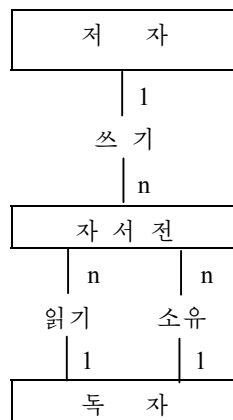



그림 11-9. 간단한 실체-관련도

이 두개가 다 1대 n관계이다. 여기서 왼쪽의 관계는 한명의 독자가 여러개의 자서전을 읽었을수 있다는 사실을 모형화하고 있다. 류사하게 오른쪽에 보여 준바와 같이 한명의 독자가 여러개의 자서전을 가질수 있다.

한명의 독자가 하나의 자서전을 소유하지 않고 읽을수 있으며 또 한명의 독자가 자서전은 사지만 그것을 읽지 않을수 있기때문에 두개의 분리된 관계들을 보여 준다. 다음의 실례는 공급자의 령역과 그들이 공급하는 부분품들로부터 얻어 진다. 그림 11-10은 부분품들과 공급자들사이의 n대 n관계를 보여 주고 있다. 즉 한명의 공급자는 여러개의 부분품을 공급하며 거꾸로 하나의 특수한 부분품이 여러 공급자들로부터 얻어 질수 있다.

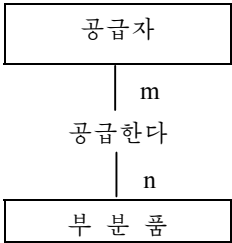



그림 11-10. n대 n (*many-to-many*) 실체-관련도

이 n대 n관계는 실체(**공급자**)의 밑에 쓴 m과 실체(**부분품**)우에 쓴 n에 의하여 반영된다.

또한 보다 복잡한 관계들도 있을수 있다. 실례로 그림 11-11에 보여 준바와 같이 이번에는 하나의 **부분품**이 많은 요소 **부분품**들로써 이루어 지는것처럼 보일수 있다.

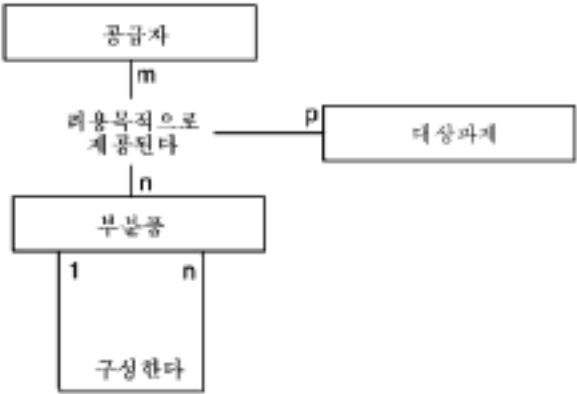



그림 11-11. 보다 복잡한 실체-관련도

또한 n대 n대 n 관계가 가능하다. 그림에서 보여 준 3개의 실체 **공급자**, **부분품**, **프로젝트**를 고찰하자.

하나의 특수한 부분품이 해당 프로젝트에 따라 여러 공급자들로부터 공급될수 있다. 또한 하나의 특수한 프로젝트에 대하여 공급되는 여러가지 부분품들은 서로 다른 공급자

들로부터 공급될수도 있다. n대 n대 n 관계는 이와 같은 상황을 정확하게 모형화하는데 필요하다.

실체-관련모형화는 다음장에서 더 론의되는데 거기에서는 객체지향분석과 또 하나의 준형식적기법을 서술한다. 이 장의 다음화제는 형식적기법들이다. 다음 4개 절의 기본문제는 형식적기법을 채용하는것이 준형식적 또는 비형식적기법으로 달성가능한것보다 더 정확하게 명세서를 작성할수 있다는것이다. 그러나 형식적기법의 리용은 일반적으로 오랜 숙련을 필요로 하며 형식적기법을 리용하는 쏘프트웨어공학자들에게는 련관된 수학적 지식을 소유하는것이 필요하다. 다음절에서 최소한도의 수학적내용을 서술하였다. 더우기 가능한 개소마다 수학적공식화를 동등한 내용의 비형식적표현에 선행하여 서술한다. 그러나 11.6부터 11.9까지의 준위는 이 책의 나머지부분들의 준위보다 더 높다.

11. 6. 유한상태기계

영국의 방송대학(*Open University*)의 M202개발팀이 처음으로 형식화한 다음과 같은 실례를 고찰하자[Brady, 1977]. 어떤 안전기는 1, 2, 3으로 표식된 3개의 위치중의 하나에 놓일수 있는 하나의 열쇠조합을 가지고 있다. 번호판은 왼쪽으로부터 오른쪽으로 돌아갈수 있다(L or R). 결국 임의의 시각에 6개의 번호판이동이 가능하다. 즉 1L, 1R, 2L, 2R, 3L, 3R이다. 안전기에 대한 열쇠조합은 1L, 3R, 2L이라고 하자. 즉 임의의 다른 번호판이동은 경고를 발생하게 할것이다. 이 상황을 그림 11-12에서 보여 주고 있다. 여기에는 하나의 초기상태 **안전보호**(Safe Locked)가 있다. 만일 입력이 1L이라면 다음상태는 A이다. 그러나 만일 임의의 다른 번호판이동 이를테면 1R와 3L이 발생하면 그다음 상태는 두개의 최종상태중의 하나인 **경고음**(Sound Alarm)으로 된다.

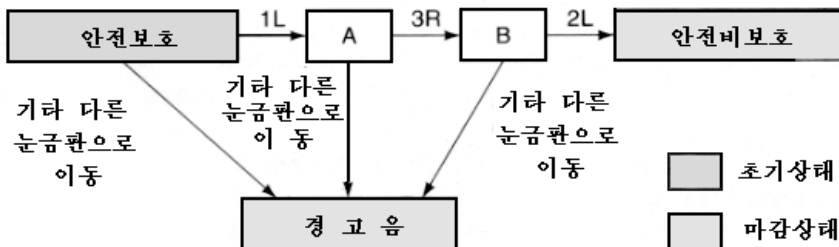



그림 11-12. 결합안전의 유한상태기계표현

만일 정확한 열쇠조합이 선택되면 이 행렬은 **안전보호**(Safe Locked)로부터 A, B, **안전비보호**(Safe Unlocked)(최종상태)로 된다.

그림 11-12는 유한상태기계의 상태이행도(*state transition diagram*; STD)를 보여 주고 있다. STD를 그라프적으로 표현하는것은 필요 없다. 이와 동일한 정보를 그림 11-13에서 표현적인 형식으로 보여 주었다. 표에서는 두개의 최종상태가 아닌 매 상태에 관하여 다음상태에로의 이행을 번호판이 움직이는 방법에 기초하여 지적하고 있다.

눈금판이동 \ 현재상태	다 음 상 태 표		
	안전보호	A	B
1L	A	경고음	경고음
1R	경고음	경고음	경고음
2L	경고음	경고음	경고음
2R	경고음	경고음	경고음
3L	경고음	경고음	경고음
3R	경고음	B	경고음

그림 11-13. 유한상태기계를 위한 상태이동표

유한상태기계는 5개의 부분들로 이루어 진다. 즉 상태들의 모임 J, 입력모임 K, 현재의 상태와 현재의 입력이 주어 진 조건에서 다음상태를 규정하는 이행함수 T, 초기상태 S 그리고 최종상태모임 F로 이루어 진다. 안전기에 대한 열쇠조합의 경우에 그 모임들은 각각 다음과 같다.

상태모임 J는 {**안전보호**, A, B, **안전비보호**, **경고음**}이다.
입력모임 K는 {1L; 1R, 2L, 2R, 3L, 3R}이다.
이행함수 T는 그림 11-13의 표형식으로 표현된다.
초기상태 S는 **안전보호**이다.
최종상태모임 F는 {**안전비보호**, **경고음**}이다.

보다 형식적인 용어로 표시하면 유한상태기계는 다음조건을 가지는 5원묶음(J, K. T, S, F)이다.

J는 유한한 비지 않은 상태모임이다.
K는 유한한 비지 않은 입력모임이다.
T는 이행함수라고 부르는 (J∼F)×K로부터 J에로의 넘기기이다.
S∈J는 초기상태이다.
F는 최종상태들의 모임 F⊆J이다.

유한상태기계방법은 콤퓨터응용에서 광범히 리용되고 있다. 실례로 매개의 차림표구동사용자대면부는 유한상태기계의 한가지 실현이다. 어떤 차림표의 현시는 하나의 상태에 대응하며 건반에서 하나의 건을 입력하거나 마우스로 하나의 아이콘을 선택하는것은 제품을 다른 상태에로 이행하도록 하는 사건과 등가이다. 실례로 주차림표가 화면에 나타났을 때 V를 입력하면 현재의 자료모임에 대하여 크기분석을 진행하게 할수도 있다. 그다음 새 차림표가 출현하며 사용자는 G, P, 또는 R를 입력할수 있다. G를 선택하면 계산결과를 그라프로 작성하게 하며 P를 선택하면 그것을 인쇄하며, R는 주차림표에로 돌아 오게 한다. 매 이행은 다음과 같은 형식을 가진다.

$$\textbf{current state} \text{ [menu] and } \textbf{event} \text{ [option selected]} \Rightarrow \textbf{next state} \tag{11.1}$$

어떤 제품을 명시하기 위하여 FSM을 확장하는 유용한 한가지 방법은 선행한 5원묶음에 여섯번째 요소 즉 술어모임 P를 추가하는것이다. 모임 P에서 매개의 술어는 제품의 대역상태 Y의 값이다[Kampen, 1987].

보다 형식적으로 이행함수 T는 (J∼F)×K×P로부터 J에로의 넘기기이다. 이행규칙은 다음과 같은 형식을 가진다.

$$\textbf{current state} \text{ and } \textbf{event} \text{ and } \textbf{predicate} \Rightarrow \textbf{next state} \tag{11.2}$$

유한상태기계는 상태와 상태들사이의 이행에 의하여 모형화될수 있는 제품을 명시하기 위한 강력한 형식화방법이다. 이러한 형식화가 실천에서 어떻게 리용되는가를 보기 위하여 이 방법을 이제 이른바 승강기문제의 변화된 형식에 적용한다. 승강기문제에 대한 배경정보에 관해서는 다음의 《알고 싶은 문제》를 보시오.

알고 싶은 문제

승강기문제는 사실상 쏘프트웨어공학의 고전적인 문제이다. 그것은 1968년에 출판된 돈 크나스의 력사적인 저서의 제1권 *The Art of Computer Programming*[Knuth, 1968]에서 처음으로 제시되였다. 이 문제는 캘리포니아공학연구소의 수학청사에 있는 단독승강기에 기초를 두고 있다. 이 실례는 공상적인 프로그람언어 MIX에서 협동부분프로그람들을 례증하는데 리용되였다.

1980년대 중엽까지 승강기문제는 n개의 승강기에로 일반화되였다. 게다가 그 풀이의 특수한 성질들이 증명되여야 하였다. 실례로 승강기가 결국 유한시간내에 도달할것이라는 문제이다. 승강기문제는 형식적명세작성언어분야에 종사하는 연구자들을 위한 문제로 되였고 모든 제안된 형식적명세작성언어들은 승강기문제에 관하여 수행되여야 하였다.

이 문제는 1986년 쏘프트웨어명세작성 및 설계에 관한 제4차 국제연구회[IWSSD, 1986]론문집의 *ACM SIGSOFT Software Engineering Notes*에서 더 폭 넓게 제시되였다. 승강기문제는 1987년 5월 캘리포니아의 몬테레이(Monterey)에서 진행된 회의에 대한 제안에서 연구자들이 리용한 다섯가지문제들중의 하나이다. 이 문제는 론문집에서 STC-IDEC [영국의 스테베나쥐(Stevenage)에 있는 표준원격 및 유선통신국]의 엔 다비스에 의하여 승강기문제(*lift problem*)로 명명되였다. 그때로부터 이 문제는 더욱더 두드러지게 되였으며 쏘프트공학에서 형식적명세작성언어를 제외한 여러가지 확장된 기법들을 보여 주는데 일반적으로 리용되였다. 이 책에서는 이 방법을 매 기법들을 례증하기 위하여 리용한다. 왜냐하면 곧 알수 있는 바와 같이 이 문제가 보기처럼 그리 간단하지 않기때문이다.

11.6.1. 승강기문제: 유한상태기계

이 문제는 다음과 같은 제약에 따라 m층사이에서 n개의 승강기를 움직이는데 필요한 론리와 관련된 문제이다.

1. 매 승강기는 m개의 단추모임을 가지고 있는데 매 층에 하나의 단추가 대응한다. 단추들은 불이 켜지며 승강기가 대응하는 층을 찾아 가게 한다. 승강기가 대응하는 층을 찾아 갔을 때 단추의 불은 꺼진다.
2. 1층과 마지막층을 제외하고 매 층은 두개의 단추를 가진다. 하나의 단추는 승강기가 올라 갈것을 요청하며 다른 하나는 승강기가 내려 갈것을 요청하는 단추이다. 이 단추들을 누르면 불이 켜진다. 승강기가 해당한 층을 찾아 간 다음 희망하는 방향으로 움직일 때 불이 꺼진다.
3. 승강기에 아무런 요청도 없을 때 승강기는 문을 닫은채로 현재의 층에 머물러 있다.

이제 제품은 확장된 유한상태기계[Kampen, 1987]를 리용하여 명시된다고 하자. 그러면 이 문제에서는 두개의 단추모임이 존재한다. 매 n개의 승강기에 m개의 단추모임이 있다. 매 단추가 매 층에 대응한다. 이 n×m개의 단추들은 승강기안에 있기때문에 그것들은 승강기단추(*elevator buttons*)로 간주된다. 그리고 매 층에 두개의 단추가 있는데 하나는 승강기상승을 요청하는것이고 하나는 승강기하강을 요청하는것이다. 이 단추들을 층단추(*floor buttons*)로 간주한다.

한개의 승강기단추에 대한 상태이행도를 그림 11-14에 보여 주었다.

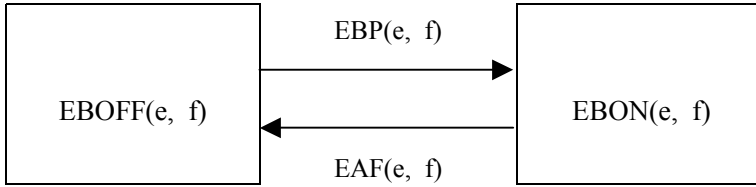



그림 11-14. 승강기단추를 위한 STD [Kampen, 1987]. (©1987 IEEE.)

EB(e, f)는 층 f를 요청하기 위하여 눌러 진 승강기 e안에 있는 단추를 의미한다고 하자. EGB(e, f)는 두가지 상태 즉 단추에 불이 온 상태든가 단추에 불이 꺼진 상태에 있을수 있다. 보다 정확하게 상태는 다음과 같다.

EBON(e, f) : Elevator Button (e, f) ON
EBON(e, f) : Elevator Button (e, f) OFF (11.3)

만일 단추가 on상태에 있고 승강기가 층 f에 이르면 단추는 off상태로 설정된다. 반대로 단추가 off상태에 있고 눌러 져 있으면 단추는 on상태로 된다.

따라서 다음의 두개의 사건이 일어 난다.

EBP(e, f) : Elevator Button (e, f) Pressed
EAF(e, f) : Elevator e Arrives at Floor f (11.4)

이 사건들과 상태들을 련결하는 상태이행규칙들을 정의하기 위하여 술어 V(e, f)가 필요하다(하나의 술어는 참 또는 거짓으로 되는 어떤 조건이다.).

$$V(e,\ f)\ :\ \underline{E}\text{levator e is }\underline{V}\text{isiting (stopped at) floor f} \tag{11.5}$$

형식적이행규칙들이 서술된다. 만일 승강기단추(e, f)가 꺼져 있고(현재상태) 승강기단추(e, f)는 눌러 지며(사건) 승강기 e는 층 f를 찾아 가지 못하고 있다(술어)면 이행규칙(11.2)의 형식으로서 이것은 다음과 같이 표현된다.

$$\text{EBOFF}(e,\ f)\ \text{and}\ \text{EBP}(e,\ f)\ \text{and not}\ V(e,\ f) \Rightarrow \text{EBON}(e,\ f) \tag{11.6}$$

만일 승강기가 현재 층 f를 찾아 가고 있다면 아무 사건도 일어 나지 않는다. 캄펜의 형식화에서 실지로 이행을 일으키지 않는 사건들이 생길수 있다. 그러나 만일 그러한 사건들이 생긴다면 그것들은 무시된다.

거꾸로 만일 승강기가 층 f에 도달하고 단추가 켜져 있다면 그것은 꺼진다. 이것은 다음과 같이 표현된다.

$$\text{EBON}(e,\ f)\ \text{and}\ \text{EAF}(e,\ f) \Rightarrow \text{EBOFF}(e,\ f) \tag{11.7}$$

이제는 층단추들을 고찰하자. FB(d, f)는 승강기가 방향 d로 움직일것을 요청하는 층 f에서의 단추를 의미한다.

층 단추 FB(d, f)를 위한 STD는 그림 11-15에 보여 주었다.

그림 11-15. 층 단추를 위한 STD [Kampen, 1987]. (©1987 IEEE.)

보다 정확하게 이 상태들은 다음과 같다.

$$\begin{aligned} &\text{FBON}(d,\ f)\ :\ \underline{F}\text{loor }\underline{B}\text{utton (d, f) }\underline{\text{ON}} \\ &\text{FBOFF}(d,\ f) :\ \underline{F}\text{loor }\underline{B}\text{utton (d, f) }\underline{\text{OFF}} \end{aligned} \tag{11.8}$$

만일 어떤 단추가 켜져 있고 승강기가 정확한 방향에로 움직여 층 f에 도착한다면 그 단추는 꺼진다.

거꾸로 만일 그 단추가 꺼져 있고 그것을 누른다면 그 단추는 켜지게 된다. 또한 다음의 두개의 사건이 일어 난다.

$$\begin{aligned} &\text{EBP}(d,\ f){:}\quad \underline{F}\text{loor }\underline{B}\text{utton (d, f) }\underline{P}\text{ressed} \\ &\text{EAF}(1..n,\ f){:}\quad \underline{E}\text{levator 1 or} \cdots \text{or n }\underline{A}\text{rrives at }\underline{F}\text{loor} \end{aligned} \tag{11.9}$$

1..n을 리용하여 분리를 표시하자. 이 절 전반에서 P(a1, .., n, b)와 같은 식은 다음의 식을 의미한다.

$$P(a,\ 1,\ b)\ \text{or}\ P(a,\ 2,\ b)\ \text{or} \cdots \text{or}\ P(a,\ n,\ b) \tag{11.10}$$

이 사건들과 상태들을 련결하는 상태이행규칙들을 정의하기 위하여 역시 술어가 필요하다. 이 경우에 그것은 S(d, e, f)를 표시하는데 다음과 같이 정의된다.

S(d, e, f): 승강기 e 는 층 f를 찾아 가고 있으며 이동하려는 방향은 우로(d=U) 혹은 아래로 (d=D) 이거나 그 어떤 요청도 (d=N)를 만족시키지 못한다. (11.11)

이 술어는 사실상 하나의 상태이다. 사실 이 형식화는 사건과 상태모두를 술어로서 취급할수 있게 한다.

그러면 S(d, e, f)를 리용하여 형식적이행규칙들을 다음과 같이 쓸수 있다.

$$\text{FBOFF}(d, f) \text{ and } \text{FBP}(d, f) \text{ and not } S(d, 1..n, f) \Rightarrow \text{FBON}(d, f) \quad (11.12)$$
$$\text{FBON}(d, f) \text{ and } \text{EAF}(1..n, f) \text{ and } S(d, 1..n, f) \Rightarrow \text{FBOFF}(d, f),\ d = U \text{ or } D$$

즉 만일 방향 d에로의 이동에 대하여 층 f에 있는 층단추가 꺼져 있고 그 단추가 눌러 져 있으며 방향 d로의 이동에 대하여 그 어떤 승강기도 현재 층 f로 찾아 가고 있지 않다면 그 층 단추는 켜진다. 거꾸로 만일 그 단추가 켜져 있고 적어도 한대의 승강기가 층 f에 도착하고 승강기는 방향 d로 움직이려 한다면 그 단추는 꺼진다. 식 S(d, 1..n, f)와 EAF(1..n, f)에서 표식 1..n은 식 (11.10)에서 정의되였다. 정의 (11.5)의 술어 V(e, f)는 S(d, e, f)에 의하여 다음과 같이 정의될수 있다.

$$V(e, f) = S(U, e, f) \text{ or } S(D, e, f) \text{ or } S(N, e, f) \quad (11.13)$$

승강기단추와 층단추들의 상태는 곧바로 정의되였다.

승강기에로 돌아 가 생각해 보면 복잡한 문제들이 생겨 난다. 승강기의 상태는 본질상 많은 요소상태들로 이루어 진다. 캄펜은 승강기의 속도감속 및 정지, 문열기, 설정시간내에서의 문열어놓기, 시간초과후의 문닫기와 같은 여러가지 상태들을 구별하고 있다 [Kampen, 1987]. 그는 승강기조종기(승강기의 이동을 지휘하는 기구)가 S(d, e, f)와 같은 상태에서 시작하며 그다음 조종기는 승강기가 부분상태들을 통해 움직인다는 론리적인 가정을 하고 있다. 세가지 승강기상태들을 정의할수 있다. 그 하나인 S(d, e, f)는 정의 (11.11)에서 정의되였는데 여기에서는 완전한 정의를 준다.

M(d, e, f): Elevator e is <u>M</u>oving in direction d(floor f is next)
S(d, e, f): Elevator e is <u>S</u>topped (d-bound) at floor f (11.14)
W(e, f): Elevator e is <u>W</u>aiting at floor f (door closed)

이 상태들은 그림 11-16에 보여 주었다. 세개의 정지상태 S(U, e, f), S(N, e, f)와 S(D, e, f)는 상태도를 간단화하기 위하여 하나의 보다 큰 상태로 묶어 졌다.

상태이행을 유발시키는 사건들은 DC(e, f), ST(e, f), RL이다. DC(e, f)는 층 f에서 승강기 e의 문을 닫는것이다. ST(e, f)는 승강기가 층 f에 가까와 갈 때 승강기수감기가 유발되여 승강기조종기가 승강기를 그 층에서 멈추어야 하는가 아닌가를 결정하여야 하는

경우에 일어 나는 사건이다. RL은 하나의 승강기단추 또는 하나의 층단추가 눌리워 그것의 ON상태를 입력할 때마다 발생한다.

DC(e, f): Door Closed for elevator e, at floor f
ST(e, f): Sensor Triggered as elevator e nears floor f (11.15)
RL: Request Logged (button pressed)

이 사건들은 그림 11-16에 보여 주었다.

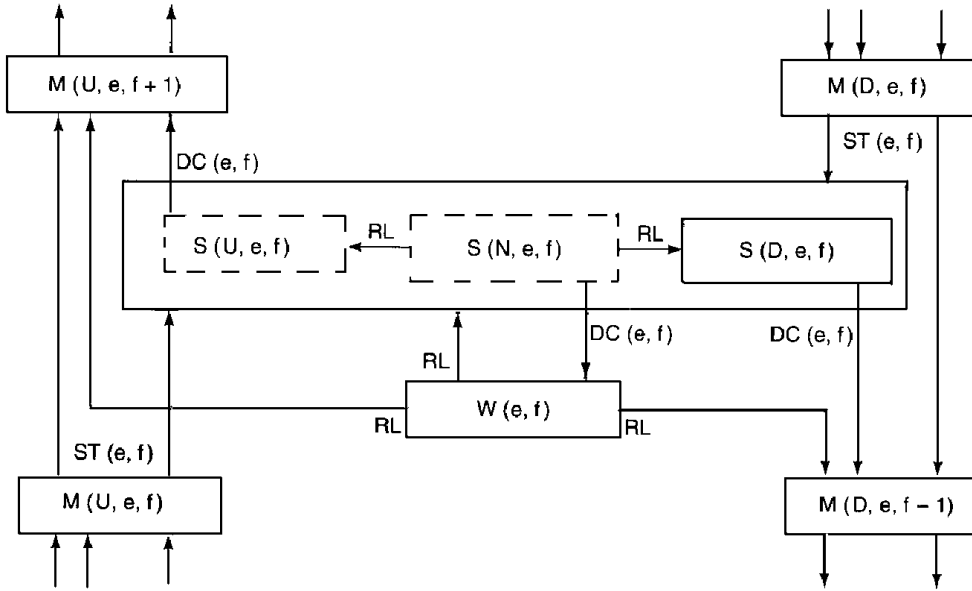



그림 11-16. 승강기를 위한 STD [Kampen, 1987]. (©1987 IEEE.)

최종적으로 하나의 승강기를 위한 상태이행규칙이 제시될수 있다. 그것들은 그림 11-6으로부터 연역될수 있지만 일부 경우에 추가적인 술어들이 필요하다. 보다 명백히 한다면 그림 11-16은 비결정론적이다.

기타 리유들로 하여 STD를 결정론적으로 하기 위한 술어들이 필요하다. 흥미 있는 독자들은 문헌 [Kampen, 1987]에 제시된 완전한 규칙모임을 참고할수 있다.

간결성을 위하여 여기서는 문을 닫을 때 무슨 상태가 발생하는가를 선언하는 규칙만을 제시한다.

승강기는 현재 이 상태에 따라서 올라 가거나 내려 가며 대기상태에 들어 간다.

$$\begin{aligned} &S(U,\ e,\ f) \ \text{ and } \ DC(e,\ f) \Rightarrow M(U,\ e,\ f+1) \\ &S(D,\ e,\ f) \ \text{ and } \ DC(e,\ f) \Rightarrow M(D,\ e,\ f-1) \\ &S(N,\ e,\ f) \ \text{ and } \ DC(e,\ f) \Rightarrow W(e,\ f) \end{aligned} \tag{11.16}$$

첫번째 규칙은 만일 승강기 e가 상태 S(U, e, f)에 있다면 즉 승강기가 올라 가려고 층 f에 멈추어 섰고 문이 닫겨 있다면 승강기는 다음층으로 올라 갈것이라는것을 서술하

고 있다. 두번째와 세번째 규칙은 승강기가 내려 가려고 하는 경우나 승강기에 대한 아무런 요청도 없는 경우에 대응한다.

이 규칙형식들은 복잡한 제품들을 명시하는데서의 유한상태기계들의 능력을 반영하고 있다.

해당 제품이 무엇을 하기 위하여 준수해야 할 복잡한 사전조건들의 모임을 렬거하고 그다음 그 제품이 그것을 수행한후에 준비해야 할 모든 조건들을 렬거하는것 대신에 명세서는 다음과 같은 간단한 형식을 취하게 된다.

current state and **event** and **predicate** ⇒ **next state**

이러한 류형의 명세서는 작성하기 쉽고 검증하기 쉬우며 설계 및 코드로 변환하기 쉽다. 사실상 유한상태기계명세서를 직접 원천코드로 번역하는 CASE도구를 구축하는것은 간단하다. 유지정비는 유한상태기계의 재연으로서 달성된다. 즉 만일 새로운 상태나 사건들이 필요하다면 명세서가 변경되며 새로운 제품판본이 직접 새로운 명세서로부터 만들어 지게 된다.

FSM방법은 11.3.1에서 제시한 게인과 사르센의 도형적기법보다도 더 명백한데 거의 그들의 방법만큼 리해하기 쉽다.

FSM방법은 한가지 결함을 가지고 있는데 그것은 대규모체계들에서 3원묶음(**state**, **event**, **predicate**)의 개수가 급격히 증가한다는것이다. 또한 게인과 사르센의 기법과 마찬가지로 캄펜의 형식화에서 시간제약은 취급하지 않고 있다. 이 문제들은 FSM의 확장인 상태도표를 리용하여 해결할수 있다[Harel et al., 1990]. 상태도표는 매우 강력한 수단이며 그것은 CASE작업프로그람인 Rhapsody에 의하여 지원되고 있다. 이 방법은 대규모실시간체계들에 성공적으로 리용되였다.

시간문제를 처리할수 있는 다른 하나의 기법은 페트리망이다.

11. 7. 페트리망

병행체계를 명시하는데서 제기되는 중요한 난점은 시간문제에 대처하는것이다. 이 난점은 동기화문제, 경쟁조건, 교착상태와 같은 여러가지 각이한 방식으로 나타난다 [Silberschatz and Galvin, 1998]. 시간문제는 비록 서투른 설계나 오유실현의 결과로서 생겨날수 있다고 하여도 이와 같은 설계나 실현은 흔히 서투른 명세작성의 결과이다. 만일 명세서가 적당하게 작성되지 않으면 대응하는 설계와 실현이 부적당하게 될수 있는 매우 실질적인 위험성이 존재하게 된다. 체계를 잠재적인 시간문제와 함께 명시하기 위한 한가지 강력한 기법은 페트리망이다. 이 방법의 또 하나의 우월성은 그것이 설계에서도 역시 리용될수 있다는것이다.

페트리망은 칼 아담 페트리에 의하여 발명되였다[Petri, 1962]. 원래 페트리는 자동체리론에 흥미를 가지고 있었다. 페트리망은 콤퓨터과학에서 광범한 응용가능성을 가지고 있는데 성능평가, 조작체계, 쏘프트웨어과학과 같은 분야들에서 리용되고 있다. 특히 페트리망은 동시에 서로 관계되는 작용들을 서술하는데서 유용하다는것이 증명되였다. 그러나 명세작성에서의 페트리망리용을 보여 주기에 앞서 그에 익숙되여 있지 않을수도 있

는 독자들을 위하여 페트리망에 대하여 간단히 소개한다.

페트리망은 4개 부분 즉 마디점모임 P, 이행모임 T, 하나의 입력함수 I, 하나의 출력함수 O로 구성된다. 그림 11-17에 보여 준 페트리망을 고찰하자.

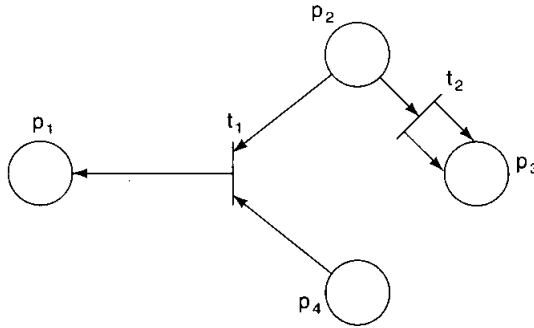



그림 11-17. 페트리망

마디점모임 P는 $\{p_1, p_2, p_3, p_4\}$이다.

이행모임 T는 $\{t_1, t_2\}$이다.

마디점으로부터 이행에로 가는 화살표들로 표시된 두개의 이행에 대한 입력함수들은 다음과 같다.

$$I(t_1) = \{p_2, p_4\}$$
$$I(t_2) = \{p_2\}$$

이행들로부터 마디점에로 가는 화살표들로 표시된 두개의 이행에 대한 출력함수들은 다음과 같다.

$$O(t_1) = \{p_1\}$$
$$O(t_2) = \{p_3, p_3\}$$

P_3이 중복되였다는것을 주목하라. 즉 t_2로부터 P_3에로 가는 두개의 화살표가 있다.

보다 형식적으로[Peterson, 1981] 하나의 페트리망은 다음과 같은 4원묶음 C=(P, T, I, O)이다.

P = $\{p_1, p_2, \cdots, p_n\}$는 마디점(*place*)들의 유한모임이다. $n \geqq 0$

T = $\{t_1, t_2, \cdots, t_m\}$는 이행(*transition*)들의 유한모임이다. $m \geqq 0$, P와 T는 분리되여 있다.

$I: T \rightarrow P^{\infty}$는 입력함수이며 이행들로부터 마디점들의 다중모임(*bag*)에로의 넘기기이다.

$O: T \rightarrow P^{\infty}$은 출력함수이며 이행들로부터 마디점의 다중모임에로의 넘기기이다(다중모임(*bag or multiset*)은 한 요소의 다중실례들을 허용하는 모임의 일반화이다.).

페트리망을 표식하는것은 그 페트리망에 대하여 표식기호들을 할당하는것이다. 그림 11-18은 4개의 표식기호(*token*)를 포함하고 있다. 즉 p_1에 1개, p_2에 2개, p_3에는 없고 p_4에

는 1개이다. 이 표식붙이기는 벡토르 (1, 2, 0, 1)로 표시할수 있다. 이행 t_1은 p_2와 p_4에 표식기호가 있기때문에 작용된다(발화될 준비가 되여 있다.). 일반적으로 하나의 이행은 만일 그 이행의 매개 입력마디점들이 그 마디점으로부터 그 이행에로 가는 호의 개수만한 표식기호들을 가진다면 작용된다. 만일 t_1이 발화되여야 한다면 p_2로부터 하나의 표식기호를 제거하고 p_4로부터 하나의 표식기호를 제거하며 하나의 새로운 표식기호가 p_1에 위치하게 된다. 표식기호들의 개수는 보존되지 않는다. 즉 두개의 표식기호가 제거되였지만 한개만이 p_1에 넣어 진다. 그림 11-18에서 이행 t_2 또한 작용된다. 왜냐하면 p_2에 표식기호들이 있기때문이다. 만일 t_2 이 발화되여야 한다면 하나의 표식기호가 p_2로부터 제거될 것이며 두개의 새로운 표식기호들이 p_3에 넣어 진다.

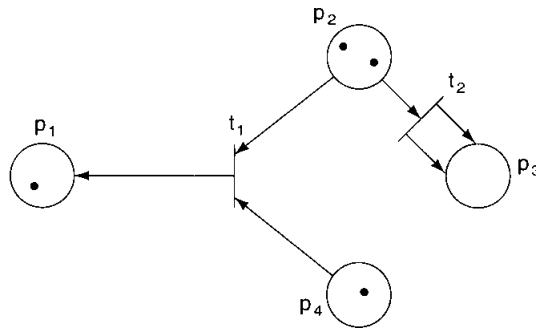



그림 11-18. 표식 붙은 페트리망

페트리망은 비결정적이다. 즉 만일 하나이상의 이행들이 발화될수 있다면 그것들 가운데서 임의의것이 발화될수 있다. 그림 11-18은 표식 (1, 2, 0, 1)을 가지고 있다. 즉 t_1과 t_2은 모두 작용된다. t_1이 발화된다고 가정하자. 결과적인 표식(2, 1, 0, 0)은 그림 11-19에 보여 주는데 여기에서는 t_2만이 작용된다. 그것이 발화되면 작용되는 표식기호는 P_2에서 제거되고 두개의 새로운 표식기호가 P_3에 놓인다. 그러면 표식은 그림 11-20에 보여 준것처럼 (2, 0, 2, 0)으로 된다.

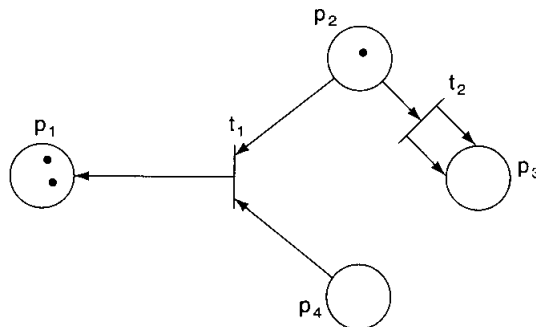



그림 11-19. 이행 t_1가 발화한 다음 그림 11-18의 페트리망

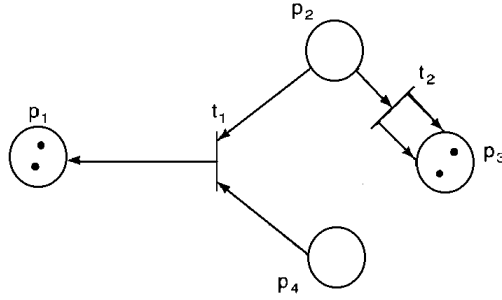



그림 11-20. 이행 t_2가 발화한 다음 그림 11-18의 페트리망

보다 형식적으로[Peterson, 1981] 하나의 페트리망 C=(P, T, O)의 하나의 표식 M은 마디점모임 P로부터 부 아닌 옹근수들의 모임에로의 넘기기이다. 즉

$$M : P \rightarrow \{0, 1, 2, \cdots\}$$

그러면 표식된 페트리망은 하나의 5원묶음(P, T, I, O, M)으로 된다.

페트리망에 대한 하나의 중요한 확장은 하나의 억제호(*inhibitor*)이다. 그림 11-21에서 억제호는 화살머리가 아니라 작은 동그라미로 표식된다.

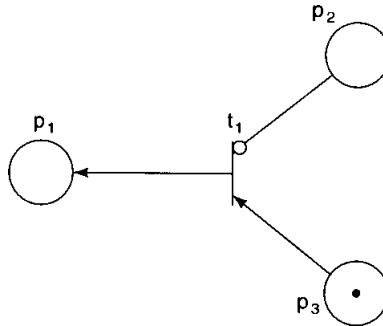



그림 11-21. 억제호를 가진 페트리망

이행 t1은 p_3에는 하나의 표식기호가 있지만 p_2에는 없기때문에 작용된다. 일반적으로 어떤 이행은 만일 그 이행의 모든(표준) 입력호들에 적어도 하나의 표식기호가 놓이고 그 이행의 어떤 억제호들에도 표식기호가 놓이지 않는다면 작용된다. 이러한 확장은 11.6.1에 제시된 승강기문제에 대한 페트리망명세작성에서 리용될것이다[Guha, Lang, Bassionui, 1987].

11. 7. 1. 승강기문제 : 페트리망

n개의 승강기체계가 m층건물에 설치되게 된다는것을 상기하자. 이에 대한 페트리망명세작성에서 건물의 매층은 하나의 마디점 F_f , 1≦f≦m으로 표시되며 페트리망에서 하

나의 승강기는 표식기호로 표현된다. F_f 의 하나의 표식기호는 한개의 승강기가 층 f에 있다는것을 의미한다.

첫번째 제약 매 승강기는 m개의 층들로 이루어 진 모임을 가지는데 매층에 하나의 단추가 대응된다. 단추들을 누르면 불이 켜지며 승강기가 대응하는 층을 찾아 가게 한다. 승강기가 대응하는 층을 찾아 가면 단추의 불은 꺼진다.

이 상황을 명세서에 넣기 위하여서는 추가적인 마디점들이 필요하다. 층 f에 대한 승강기단추는 페트리망에서 마디점 EB_f, $1 \leq f \leq m$으로 표현된다. 보다 정확히는 n개의 승강기가 있기때문에 마디점은 $EB_{f, e}$, $1 \leq f \leq m$, $1 \leq e \leq n$으로 표현되여야 하며 승강기를 나타내는 첨수 e는 생략된다.

EB_f안의 하나의 표식기호는 층 f에 대한 승강기단추가 켜졌다는것을 의미한다. 단추는 그 단추가 처음 눌리울 때만 켜지고 련이은 단추누르기는 무시되기때문에 이것을 그림 11-22에 보여 준바와 같이 페트리망을 리용하여 명시한다. 먼저 단추 EB_f가 불이 켜지지 않았다고 가정하자. 따라서 마디점안에는 아무런 표식기호도 없으며 억제호가 존재하기때문에 이행 EB_f pressed가 작용된다. 이제 그 단추를 누른다. 그 이행이 발화되며 그림 11-22에 보여 준것처럼 EB_f에 새로운 표식기호가 놓인다. 이제 단추를 아무리 많이 눌러도 억제호와 표식기호존재의 조합은 이행 EB_f pressed가 작용될수 없다는것을 의미한다. 그러므로 하나이상의 표식기호는 EB_f안에 놓일수 없다. 승강기가 층 g로부터 층 f로 움직인다고 가정하자.

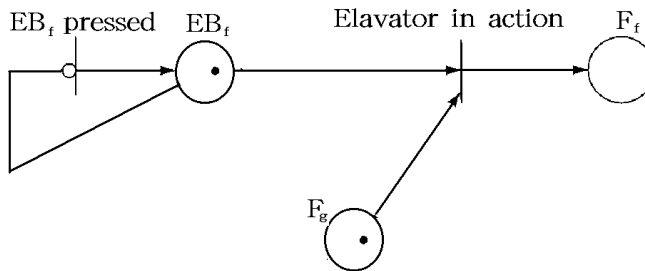



그림 11-22. 승강기단추의 페트리망표현
[Guha, Lang, and Bassiouni, 1987]. (©1987 IEEE.)

승강기는 층 g에 있기때문에 그림 11-22에 보여 준것처럼 마디점 Fg에 하나의 표식기호가 놓인다. 이행 Elevator in action이 작용되여 발화된다. EB_f와 F_g안의 표식기호들이 제거되고 단추 EB_f는 꺼지며 새로운 하나의 표식기호가 F_f안에 나타난다. 이 이행의 발화는 승강기가 층 g로부터 층 f에로 움직이게 한다.

층 g로부터 층 f에로의 이러한 이행은 동시에 일어 날수 없다. 이것과 어떤 단추를 누를 때마다 그 단추가 불이 켜질 물리적불가능성과 같은 류사한 론점들을 다루기 위하여 시간조절기능이 페트리망모형에 추가되여야 한다. 즉 고전적페트망리론과는 반대로 승강기문제와 같은 실천적정황에서 이행들은 동시적이며 하나의 령 아닌 시간을 하나의 이행과 관련시키기 위하여 시한페트리망(*timed Petrinet*)들이 필요하다[Coolahan and Raussopoulos,

1983].

두번째 제약 1층과 마지막층을 제외하고 매층은 두개의 단추를 가진다. 한개는 승강기상승을 요청하며 다른 한개는 승강기하강을 요청한다. 이 단추들을 누를 때 불이 켜진다. 승강기가 대응하는 층을 찾아 가고 그다음 요구하는 방향으로 움직일 때 그 단추의 불은 꺼진다.

FB_f^u 층단추들은 마디점 FB_f^u 와 FB_f^d 들로 표시되는데 이것들은 각각 승강기상승 및 하강을 요청하는 단추들을 표시하고 있다. 보다 정확하게 말하면 층 1은 단추 FB_1^u 를 가지며 층 m은 단추 FB_m^d 를 가지며 매 중간층들은 두개의 단추 FB_f^u 와 FB_f^d $1<f<m$를 가진다. 하나의 승강기가 하나 또는 두개의 단추가 불이 켜진 층 g로부터 층 f에 도달할 때의 정황을 그림 11-23에 보여 주었다. 사실 이 그림은 더 정교하게 묘사하여야 한다. 왜냐하면 만일 단추 두개가 다 불이 켜지면 비결정론적기준에 의하여 하나는 꺼지기때문이다.

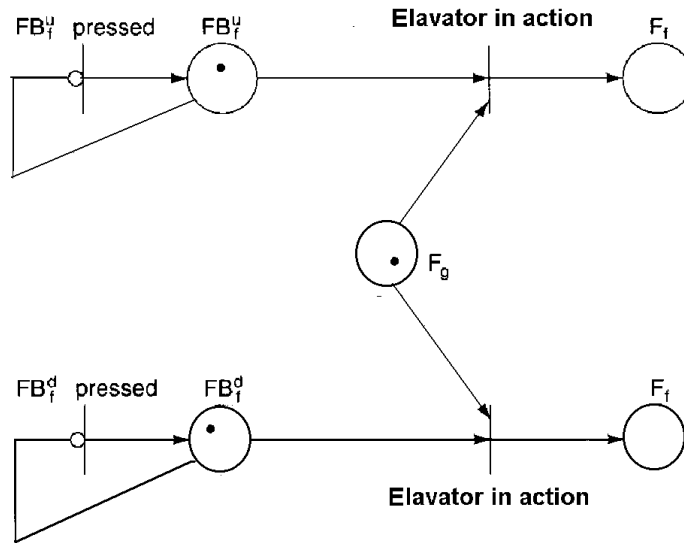



그림 11-23. 층단추의 페트리망표현
[Guha, Lang, and Bassiouni, 1987]. (©1987 IEEE.)

단추가 정확히 꺼진다는것을 확인하기 위하여서는 너무 복잡한 페트리망이 필요하기 때문에 여기서는 제시하지 않는다. 실례로 문헌 [Ghezzi and Mandnoli, 1987]을 보시오.

세번째 제약 승강기에 아무런 요청도 없을 때 승강기는 문을 닫은채로 현재의 층에 머물러 있는다.

이것은 쉽게 달성할수 있다. 즉 만일 아무런 요청도 없으면 아무런 Elevator in action 이행도 작용되지 않는다.

페트리망은 명세서를 표현하기 위하여 리용될뿐아니라 설계를 위해서도 리용될수 있다[Guha, Lang, and Bassiouni, 1987]. 제품개발의 명세작성단계에서만 보아도 페트리망이

병행체계의 동기화측면을 명시하기 위하여 필요한 표현능력을 가지고 있다는것은 명백하다.

11. 8. Z

현재 광범한 인기를 끌고 있는 한가지 형식적명세작성언어는 Z이다[Spivey, 1992](Z의 이름을 정확히 발음하기 위하여서는 다음의 《알고 싶은 문제》를 보시오.). Z를 리용하기 위하여서는 1계 론리를 비롯하여 모임론, 함수, 리산수학에 대한 지식이 필요하다. 필요한 배경지식(대부분의 콤퓨터과학전공과목들)을 가진 사용자들조차 Z는 처음에 배우기 어렵다. 왜냐하면 Z는 ∃, ⊃, ⇒와 같은 일반적인 모임론적 및 론리적기호들외에 ⊕, ⩤, ↦, ⤀와 같은 잘 쓰이지 않는 특수기호들도 리용하기때문이다.

알고 싶은 문제

형식적명세작성언어에 대한 Z라는 이름은 그 창시자 진 레이몬드 아브리알(Jean-R ymond Abrial)이 뛰여난 모임론학자인 에른스트 프레드리츠 페르디난드 제르멜로(Ernst F iedrich Ferdinand Zermelo)(1871-1953)에게 경의를 표시하여 지었다. 이것은 옥스포드대학‹ 서 개발되였기때문에[Abrial, 1980] Z라는 이름을 영어자모 26번째를 영국식으로 발음하- · 방식으로《zed》라고 적당히 발음한다.

그러나 최근에 Z가 독일수학자의 이름을 따서 지었다는것을 인정하고 그것을 독‹ 식으로《tzet》라고 발음하려는 움직임이 보이고 있다. 이에 대응하여 프란코휠레스(Francop iles)와 프란코호네스크(Francophonesk)는 아브리알이 프랑스사람이라는것과 문자 Z가 프링스식으로 《zed》라고 발음된다는것을 지적하였다.

총적으로 수용할수 없는 한가지 발음은 미국식으로 그것을 《zee》로 발음히 는것이다. 그 리유는 Z(《zee)라고 발음)가 4세대프로그람작성언어의 이름이기때문이다(1 .2) 그러나 영어자모의 한개 문자에 상표를 붙일수 없다. 더우기 우리는 문자 Z를 원하- · 방식으로 자유롭게 발음하고 있다. 그러나 프로그람작성언어의 범위내에서 《zee》라- · 발음은 4GL과 관계되여 있고 형식적명세작성언어와는 관계되지 않는다.

Z가 제품을 명시하는데 리용되는 방법을 파악하기 위하여 11.6.1의 승강기문제를 다시 고찰하자.

11. 8. 1. 승강기문제: Z

가장 간단한 형식으로 Z는 다음의 네 부분으로 구성된다.

1. 주어 진 모임, 자료류형, 상수
2. 상태정의
3. 초기상태

4. 조작들

이 매 부분들을 차례로 고찰하자.

1. 주어 진 모임(Given Sets) Z명세서는 주어 진 모임들의 목록 즉 세부적으로 정의될 필요가 없는 모임들로 시작한다. 이와 같은 임의의 모임들의 이름은 중괄호안에 나타난다. 승강기문제에서 주어 진 모임은 모든 단추들의 모임인 Button으로 불리울것이다.

그러므로 Z명세서는

[Button]

으로 시작한다.

2. 상태정의 Z명세서는 많은 도식(*schema*)들로 구성된다.

매 도식들은 변수들의 가능한 값들을 제한하는 술어들의 목록과 함께 변수선언들의 그룹으로 이루어 진다. 도식 S의 형식을 그림 11-24에 보여 주었다.

```
———————— S ————————————————
| declarations
|———————————————————————————
| predicates
|———————————————————————————
```

그림 11-24. Z도식 S의 형식

승강기문제에서는 Button에 대한 4개의 부분모임이 존재한다. 즉 층단추들, 승강기단추들, buttons(승강기문제에서 모든 단추들의 모임)와 pushed(눌리워 진 단추들의 모임)이다. 그림 11-25는 도식 *Button_state*를 묘사하고 있다. 기호 P는 제곱모임(주어 진 모임의 모든 부분모임들의 모임)을 의미한다.

제약들 즉 수평선아래의 설명문들은 floor_buttons와 elevator_buttons모임들이 분리되여 있으며 그것들이 합쳐 져서 buttons모임을 구성한다는것을 서술하고 있다(floor_buttons와 elevator_buttons모임들은 이후에는 필요 없다. 즉 그것들은 Z의 능력을 보여 주기 위하여만 그림 11-25에 포함된다.).

```
———————— Button_State ————————————————————————————
| floor_buttons, elevator_buttons   :P Button
| buttons                           :P Button
| pushed                            :P Button
|——————————————————————————————————————————————————
| floor_buttons ∩ elevator_buttons = Ø
| floor_buttons ∪ elevator_buttons = buttons
|——————————————————————————————————————————————————
```

그림 11-25. Z도식*Button_State*

3. 초기상태 추상적초기상태(*abstract initial state*)는 체계가 처음 시동될 때의 상태를

서술한다. 승강기문제에 대한 추상적초기상태는 다음과 같다.

$$Button_Init \triangleq [Button_State' \mid \mathrm{pushed}' = \varnothing]$$

이것은 수직형도식정의(*vertical schema definition*)이며 그림 11-25에 보여 준것과 같은 수평구도정의(*horizontal schema definition*)와 반대이다. 수직형도식은 승강기가 처음 시동하였을 때 모임 pushed는 초기에 비여 있다는것 즉 모든 단추들이 꺼져 있다는것을 서술하고 있다.

4. 조작 어떤 단추를 처음 누르면 그 단추에 불이 켜진다. 그 단추는 모임 pushed에 첨부된다. 이것은 그림 11-26에 보여 주고 있는데 거기에서 조작 *Push_Button*이 정의된다. 도식의 첫행에 있는 △는 이 조작이 *Button_State*의 상태를 변화시킨다는것을 의미하고 있다. 조작은 하나의 입력변수 button?을 가지고 있다. 기타 여러가지 언어(CSP[Hoare, 1985]와 같은)에서와 마찬가지로 의문기호(?)는 입력변수를 의미하며 감탄표(!)는 출력변수를 의미한다.

한 조작의 술어부분은 그 조작이 호출되기전에 준수하여야 할 사전조건들과 그 조작이 실행을 끝낸후에 준수하여야 할 사후조건들의 그룹으로 이루어 진다. 사전조건들이 만족된다는 조건하에서 사후조건들은 실행을 완성한후에 준수될것이다. 그러나 만일 그 조작이 사전조건이 만족됨이 없이 호출된다면 명시되지 않은(그러므로 서술할수 없는) 결과들이 발생할수 있다.

그림 11-26. 조작 *Push_Button*의 Z서술

그림 11-26의 첫번째 사전조건은 button?가 이 승강기체계의 모든 단추들의 모임인 buttons의 성원이여야 한다는것을 서술하고 있다. 만일 두번째 사전조건 button?∉pushed가 만족된다면(즉 만일 그 단추가 켜져 있지 않다면) pushed단추들의 모임은 갱신되여 button?을 포함한다. Z에서 어떤 변수의 새로운 값은 빗점(′)으로 표시된다. 결국 사후조건은 조작 *Push_Button*이 수행된 다음에 button?이 모임 pushed에 추가되여야 한다는것을 서술하고 있다. 명백히 그 단추에 불을 켤 필요는 없다. button?이 이제는 pushed의 요소이라는것은 충분하다.

또 다른 가능성은 이미 눌러 진 단추를 또 누르는것이다. button?∈pushed이기때문에 세번째 사전조건은 성립[*)]되며 필요할 때 아무런 사건도 발생하지 않는다. 이것은 서술문

[*)] 세번째 사전조건이 없는 경우 명세서는 만일 이미 누른 단추를 다시 누르면 어떤 사건이 일어 나겠는가를 서술할수 없을것이다. 그렇게 되면 결과들을 명시할수 없을것이다.

pushed'= pushed로 지적된다. 즉 이것은 pushed의 새 상태는 낡은 상태와 갈다는것을 의미한다.

이제 승강기가 어떤 층에 도착하였다고 가정하자. 만일 대응하는 층 단추가 켜져 있다면 그것은 꺼져야 하며 대응하는 승강기단추에 대해서도 이것은 류사하다. 즉 만일 button?이 pushed의 요소이라면 그림 11-27에 보여 준바와 같이 그림은 이 모임으로부터 제거되여야 한다(기호 \은 모임차를 의미한다.). 그러나 만일 어떤 단추가 켜져 있지 않다면 모임 pushed는 변화되지 않는다.

Floor_Arrival

$$\Delta Button_State$$
$$button?: Button$$

$$(button? \in buttons) \wedge$$
$$(((button? \in pushed) \wedge (pushed' = pushed \setminus \{button?\})) \vee$$
$$((button? \notin pushed) \wedge (pushed' = pushed)))$$

그림 11-27. 조작 *Floor_Arrival*의 Z서술

이 절에서 제시한 해결방법은 지나치게 단순화한 한가지 실례이다. 즉 여기에서는 승강기상승 및 하강단추를 구분하지 않고 있다. 그러나 Z가 승강기문제에서 단추들의 거동을 명시하기 위하여 어떻게 리용될수 있는가 하는 하나의 지시를 주고 있다.

11. 8. 2. Z의 분석

Z는 CASE도구들[Hall, 1990], 실시간핵심부[Spivey, 1990], 오씰로그라프[Delisle and Garlan, 1990]를 비롯한 광범한 프로젝트들에 성공적으로 리용되였다.

Z는 또한 IBM업무처리체계인 CICS배포판의 큰 부분을 명시하는데 리용되였다[Nix and Collins, 1988]. 이와 같은 성공은 승강기문제와 같은 단순화된 경우에조차 Z가 똑바로 리용되지 않는다는 견지에서 보면 좀 놀라운 일이다. 첫째로, 표시법에서 문제가 발생한다. 즉 새로운 사용자는 Z명세서를 읽을수 있고 그것을 단독으로 작성할수 있게 되기에 앞서 기호모임과 그 의미를 학습하여야 한다. 둘째로, 매 쏘프트웨어공학자들이 Z를 리용할수 있는 그런 필요한 수학적숙련이 없다는것이다(비록 거의 모든 콤퓨터과학과정안을 마친 최근 졸업생들이 Z의 리용과 관련한 충분한 수학을 알고 있거나 그들이 알아야 할 필요가 있는것을 쉽게 배울수 있다 하여도 그렇다.).

Z는 아마도 그와 같은 류형의 언어들가운데서 가장 광범히 리용된 형식언어이다. 이것은 무엇때문인가? 왜 Z가 그토록 성공적인가? 특히는 대규모프로젝트에서 성공적인가? 여러가지 각이한 리유들이 제시되였다.

1. Z로 작성한 명세서들에서 특히 명세서자체의 검토와 설계나 코드의 검토기간에 오유를 찾아 내는것이 형식적명세서에 비하여 쉽다는것이 밝혀 졌다[Nix and

Collins, 1988; Hall, 1990].

2. Z명세서를 작성하는것은 명세작성자가 극히 정확할것을 요구한다. 즉 이러한 정확성에 대한 요구의 결과로서 Z명세서에는 비형식적명세서들보다 애매성, 모순, 생략이 더 적게 나타난다.
3. 형식언어로서 Z는 개발자가 필요할 때 명세서의 정확성을 증명할수 있게 한다. 결국 비록 일부 쏘프트웨어개발기업체들은 Z의 정확성증명을 거의 할수 없다고 하여도 CICS기억관리자와 같은 실천적문제들에서까지 이와 같은 증명이 진행되였다[Woodcock, 1989].
4. 고등학교 수학지식만을 가진 쏘프트웨어전문가들이 상대적으로 짧은 시간내에 Z명세서를 작성하도록 가르칠수 있다는것이 제기되였다[Hall, 1990]. 명백히 이러한 개별적사람들은 결과적인 명세서가 정확하다는것을 증명할수는 없지만 형식적명세서는 그 정확성을 반드시 증명하여야 하는것이 아니다.
5. Z의 리용은 쏘프트웨어의 개발비용을 감소시켰다. Z명세서자체에는 비형식적기법을 리용할 때보다 더 많은 시간이 소비되여야 한다는것을 의심할 여지가 없지만 완전한 개발공정에 소비되는 총적시간은 감소된다.
6. 의뢰자가 Z로 쓴 명세서를 리해할수 없다는 문제는 그 명세서를 자연언어로 다시 쓰는것을 비롯한 여러가지 방법으로 해결되였다.

결과적인 자연언어명세서는 처음부터 작성된 비형식적명세서보다 더 명백하다는것이 밝혀 졌다(이것은 또한 11.2.1에서 서술된 나우르의 본문처리문제에 관하여 메이어의 형식적명세서에 있는 영어단락으로부터 얻은 경험이다.).

가장 중요한것은 모순되는 론의에도 불구하고 Z가 많은 대규모프로젝트들에 관하여 쏘프트웨어산업에서 성공적으로 리용되였다는것이다. 대다수의 명세서들이 비록 Z보다는 매우 덜 형식적인 언어로 계속 작성된다고 하여도 형식적언어의 리용을 지향하는 대역적추세는 증대되고 있다. 이와 같은 형식적명세서는 전통적으로 유럽의 실천에서 많이 리용되여 왔다. 그러나 점점 더 많은 쏘프트웨어개발기업체들이 한가지 또는 기타 형식적명세작성법을 받아 들이고 있다. Z와 그와 류사한 언어들이 앞으로 어느 정도 리용되겠는가는 두고 보아야 한다.

11. 9. 기타 형식적기법

기타 많은 형식적기법들이 제안되였다. 이 기법들은 매우 다양하다. 실례로 Anna[Luckham and von Henke, 1985]는 Ada용형식적명세서작성언어이다. 일부 형식적기법들은 Gist[Balzer, 1985]와 같은 지식에 기초한 기법들이다. Gist는 우리들이 공정에 대하여 생각하는 방법과 될수록 가까운 방법으로서 사용자들이 공정을 서술하도록 자기의 기법을 설계하였다. 이것은 자연언어에서 리용된 구조들을 형식화함으로써 달성할수 있었다. 사실 Gist의 명세서는 기타 대다수 명세서들처럼 Gist명세서로부터 영어로 의역하는 의역기가 작성될 정도로 리해하기 어렵다.

윈정의방법(*Vienna definition method*; VDM)[Jones, 1986b]은 외연적의미론에 기초한 기법이다[Gordon, 1979]. VDM은 명세작성에뿐아니라 설계와 실현단계에도 적용될수 있다.

VDM은 많은 프로젝트들에서 성공적으로 리용되였다. 특히 DDC(Dansk Datamatik Center) Ada콤파일러체계의 개발에서 가장 성공적으로 리용되였다[Oedt, 1986].

명세서를 고찰하는 다른 한가지 방법은 그것들을 사건들의 순서렬로 간주하는것이다. 여기서 하나의 사건은 하나의 단순한 작용이든가 자료를 체계안으로 또는 밖으로 전달하는 하나의 정보전달이다. 실례로 승강기문제에서 하나의 사건은 승강기 e에서 층 f에 대한 승강기단추를 누르고 그 결과 그 단추에 불이 켜지는것으로 구성된다. 또 다른 사건으로는 승강기 e가 아래방향을 향하여 층 f를 떠나고 대응하는 층단추의 불이 꺼지는것을 례로 들수 있다. 호아(Hoare)가 발명한 언어통신순차과정(*Communicating Sequential Processes*; CSP)은 이와 같은 사건들로서 체계의 기동을 서술하려는 착상에 기초하고 있다[Hoare, 1985]. CSP에서 하나의 공정은 그 공정이 관계하게 될 환경에서의 사건들의 렬로서 서술된다. 공정들은 통보들을 서로 보내는것으로서 호상작용한다. CSP는 공정들이 순차적으로, 병렬로, 비결정론적으로 분리되는것과 같이 다양한 방법으로써 결합되도록 한다.

CSP의 능력은 CSP명세서의 실행가능한 성질에 있다[Delisle and Schwartz, 1987]. 따라서 명세서들의 내부적일치성을 검사할수 있다. 더우기 CSP는 매 단계들에서 타당성을 유지하면서 명세작성으로부터 설계에로 또 실현에로 이행하기 위한 틀거리(*framework*)를 제공하여 준다. 달리 말하면 만일 명세서들이 정확하고 변환이 정확하게 수행된다면 설계와 실현 역시 정확하게 진행될것이다. 만일 실현언어가 Ada이라면 설계로부터 실현에로 이행하는것은 매우 간단하다.

그러나 CSP도 자기의 약점을 가지고 있다. 특히 Z와 마찬가지로 그것을 배우기가 쉽지 않다. 승강기문제에 대한 CSP명세서[Schwartz and Delisle, 1987]를 이 책에 포함시키려고 시도하였었다. 그러나 본질적인 예비자료들의 질과 매 CSP서술을 적절하게 서술하는데 필요한 설명의 세부준위가 너무 방대하여 이 책과 같은 일반적인 한 권의 책에 그것을 포함시키기는 어렵다. 명세작성언어의 능력과 그 리용의 어려운 정도사이의 관계에 대하여서는 다음절에서 설명한다.

11. 10. 명세작성기법들의 비교

이 장에서의 중요한 교훈은 매 개발기업체가 어느 류형의 명세작성언어가 개발될 제품에 적합한가를 결정하여야 한다는것이다. 비형식적기법을 배우기는 쉽지만 준형식적인 또는 형식적인 기법이 가지고 있는 능력이 결여되여 있다. 거꾸로 매개의 형식적기법들은 각이한 특성들을 지원하고 있는데 이러한 특성들에는 실행가능성, 정확성증명, 단계별 정확성을 유지하면서 설계 및 실현에로 변환하는 가능성들이 포함된다. 일반적으로 그 기법이 보다 형식적일수록 그 능력은 더욱 커지며 형식적기법들을 리용하고 배우기가 어려울수 있다. 또한 형식적명세서는 의뢰자가 리해하기 어려울수 있다. 달리 말하면 명세작성언어의 리용과 능력사이에는 절충을 하여야 한다.

일부 정황에서 명세작성언어의 류형을 선택하는것은 쉽다. 실례로 만일 대다수의 개발팀성원들이 콤퓨터과학에 대하여 아무런 파악도 없다면 비형식적 또는 준형식적명세작성기법이 아닌 다른 기법을 리용하는것은 사실상 불가능하다. 거꾸로 기업에 사활적의의를 가지는 실시간체계를 실험실에서 구성하고 있다면 여기에서는 형식적명세작성기법이 거의 틀림없이 요구될것이다.

추가적인 한가지 복잡한 인자는 대부분의 보다 새로운 형식적기법들은 실천적조건에서 검증되지 않았다는것이다. 이러한 기법을 리용할 때 상당한 위험성이 포함된다. 관련있는 개발팀성원들을 양성하는데 막대한 자금을 투자해야 하며 또한 개발팀이 연구실에서 그 언어를 리용하는것으로부터 실지프로젝트에서 리용하는것을 조종하는 기간에 보다 많은 자금이 소비되는것이다. 더우기 그 언어들이 지원하고 있는 쏘프트웨어도구들이 SREM에서처럼 적당하게 동작하지 않을수도 있으며[Scheffer, Stone, and Rzepka, 1985] 결국 추가적인 지출과 기한어김이 초래될수 있다.

그러나 만일 모든것이 동작하며 쏘프트웨어의 유지정비계획에서 새로운 기법을 특수한 프로젝트에 처음으로 리용할 때 필요되는 추가적인 시간과 자금을 고려한다면 많은 리득이 차례질수 있다.

서술방법	범주	우 점	약 점
자연언어(11.2)	비형식적	배우기 쉽다. 리용하기 쉽다. 의뢰자가 리해하기 쉽다.	부정확성 설계명세가 애매하고 모순이 있고 그리고/혹은 불완전하다.
실체-관련모형화(11.5) PSL/PSA(11.4) SADT(11.4) SREM(11.4) 구조화체계분석(11.3)	준형식적	의뢰자는 비형식적인 방법보다 더 정확히 리해할수 있다.	형식적방법만큼 정확치 못하다. 일반적으로 시간을 조종할수 없다.
Anna(11.9) CSP(11.9) 확장된 마감상태기계(11.6) 요점(11.9) 페트리망(11.7) VDM(11.9) Z(11.8)	형식적	아주 정확하다. 설계명세오유를 줄일수 있다. 개발비용과 노력을 줄일수 있다. 정확성증명을 지원할수 있다.	배우기가 힘들다. 리용하기 힘들다. 대부분 사용자들이 거의나 리해할수 없다.

그림 11-28. 이 장에서 론의한 명세작성방법의 개요와 해당한 절

이런 특정한 프로젝트에 어느 명세작성기법들을 리용하여야 하는가? 이것은 프로젝트, 개발팀, 경영팀 그리고 의뢰자가 어떤 특정한 방법을 리용하겠다고(또는 리용하지 않겠다고) 고집하는것과 같은 무수히 많은 다른 인자들에 의존하고 있다. 쏘프트웨어공학의 많은 측면들에서와 마찬가지로 절충을 하여야 한다. 그러나 유감스럽게도 어느 명세작성기법을 리용하겠는가를 결정하기 위한 그 어떤 단순한 규칙도 없다.

그림 11-28은 이 절의 착상에 대한 종합이다.

11. 11. 명세작성단계에서의 시험

명세작성단계에서 목적하는 제품의 기능성이 명세서에 명확히 표현된다. 명세서가 정확한가를 검증하는것은 사활적인 문제이다. 이것을 위한 한가지 방도는 명세서관통심사회의에 의거하는것이다(6.2.1).

명세서에서 오유를 발견하기 위한 보다 강력한 수단은 검토이다(6.2.3). 검토팀은 검사목록과 대비하여 명세서를 심사한다. 명세검사목록의 전형적인 항목들에는 다음과 같은 것들이 포함된다.

필요한 하드웨어자원들이 명시되였는가?

수용판정기준이 명시되였는가?

설계와 코드의 검토와 관련하여 파간(Fagan)이 처음으로 검토(*inspection*)방법을 제안하였다[Fagan, 1976]. 파간의 론문은 6.2.3에 자세히 서술되여 있다. 그러나 검토는 명세서를 검토하는데도 상당히 쓸모 있다는것이 증명되였다. 실례로 둘렌(Doolan)은 2백만행이상의 FORTRAN언어로 구성된 제품명세서를 검증하는데 검토를 리용하였다[Doolan, 1992]. 제품에 있는 오유를 수정하기 위한 비용에 대한 자료로부터 그는 검토에 투자된 매 한시간이 실행에 기초한 오유검사와 정정에 드는 30h을 절약한다는것을 추론해 내였다.

명세서가 형식적기법을 리용하여 작성되였을 때 다른 시험기법들이 리용될수 있다. 실례로 정확성증명방법들(6.5)이 도입될수 있다. 만일 형식적증명이 진행되지 않는다고 하여도 6.5.1에서 리용된것과 같은 비형식적증명기법들은 명세서의 오유를 밝혀 내기 위한 매우 유용한 방법으로 될수 있다. 사실 제품개발은 그것의 증명과 병렬로 진행되여야 한다. 이런 방법으로 오유들은 신속히 발견된다.

11. 12. 명세작성단계에서의 CASE도구

명세작성단계에서 두가지 부류의 CASE도구들이 특히 도움이 된다. 첫째는 도형도구이다. 어떤 제품이 자료흐름도, 페트리망, 실체-관련도를 리용하여 표현되든지 아니면 지면상리유로 하여 이 책에서 언급하지 않은 기타 여러가지 방법으로 표현되든지간에 전체적인 제품을 수동적으로 작성하는것은 시간이 많이 드는 공정이다. 더우기 부분적인 변경을 진행하는것이 모든것을 처음부터 재작성해야 할 결과를 초래할수도 있다. 그렇기때문에 그리기도구는 시간을 크게 절약하게 한다. 명세서를 위한 기타 여러가지 도형표현들이 있는것처럼 이러한 류형의 도구는 이 장에서 서술되는 명세작성기법에도 있다. 명

세작성단계에서 요구되는 두번째 도구는 자료사전이다. 5.4에서 언급한것처럼 이 도구는 자료흐름과 그 요소들, 자료저장과 그 요소들, 처리(작용)와 그 내부변수들을 비롯하여 제품의 매 자료항목의 매 요소들의 이른바 표현(형식)들을 저장하고 있다(그림 11-5는 쎌리의 쏘프트웨어상점을 위한 자료사전에 기억될 전형적인 정보들을 보여 주고 있다.). 또한 자료사전의 많은 부분이 각이한 하드웨어상에서 운영된다.

실지로 필요한것은 개별적인 도형도구와 개별적인 자료사전이 아니다. 그대신 이 두 도구는 통합되여야 한다. 그리하여 자료요소에 대하여 진행한 임의의 변경은 명세서의 대응하는 부분에서 자동적으로 반영되도록 하여야 한다. 이러한 류형의 도구들에 대한 실례들로는 분석기/설계기(*Analyst*/*Designer*), 그림을 통한 쏘프트웨어(*Software through Picture*), 체계구성(*System Architecture*)들을 들수 있다. 더우기 이러한 여러가지 도구들에는 명세서와 대응하는 설계문서사이의 일관성을 검사하는 자동일관성검사기가 병합되여 있다. 실례로 명세서안의 매 항목이 설계문서에 선행하여 수행되며 설계에 언급된 모든것이 자료사전에 선언되였다는것을 검사할수 있다. 명세작성기법은 그 기법을 지지하는 풍부한 도구를 가진 CASE환경이 없으면 광범하게 리용될수 없을것이다. 실례로 SREM(11.4)은 아마도 현재 REVS보다도 훨씬 광범히 리용되는것 같은데 그와 관련된 CASE도구모임은 미공군에서의 시험을 보다 훌륭히 수행하였다[Scheffer, Stone, Rzepka, 1985]. 경험 있는 쏘프트웨어전문가들에 있어서도 어떤 체계를 정확하게 명시하는것은 쉬운 일이 아니다. 명세작성자들에게 가능한 모든 방법으로 그들을 도와 주는 최첨단 CASE도구들의 모임을 제공하는것만이 합리적일것이다.

11.13. 명세작성단계에서의 척도

명세작성기간에 다른 모든 단계들과 마찬가지로 5개의 기본척도 즉 크기, 비용, 기간, 로력, 품질을 측정하는것이 필요하다. 어떤 명세서의 크기에 관한 한가지 척도는 명세서의 페지수이다. 만일 여러가지 류사한 제품들을 명시하기 위하여 같은 기법을 리용한다면 명세서크기에서의 차이는 여러가지 제품을 만드는데 필요한 로력에 대한 중요한 예측인자로 될수 있다.

품질문제를 고찰하면 명세서검토의 한가지 중요한 측면은 오유통계에 대한 기록이다. 검토기간에 발견한 매 류형의 오유개수를 적어 놓는것은 검토공정에서 필수적인 부분이다. 또한 오유검출률도 검토공정의 효과성에 대한 척도를 줄수 있다.

목표제품의 크기를 예측하기 위한 척도는 자료사전에 들어 있는 항목개수들을 포함한다. 파일, 자료항목, 처리(작용)의 개수를 비롯하여 여러가지 각이한 계수가 진행될수 있다. 이러한 정보는 그 제품을 만드는데 요구되는 로력을 고려하여 관리자측에게 초보적인 평가를 줄수 있다. 이 정보는 기껏해서 림시적이라는것을 주의해 두어야 한다. 결국 설계단계에서 DFD(자료흐름도)안의 하나의 처리를 서로 다른 여러개의 모듈안에 갈라 넣을수 있다. 반대로 여러개의 처리들을 묶어 하나의 모듈을 구성할수도 있다. 그러나 자료사전으로부터 유도된 척도는 관리자측에게 목표제품의 최종크기에 관한 하나의 조기 실머리를 줄수 있다.

11. 14. 항공음식전문회사실례연구 : 구조화체계분석

요구사항확정단계에서 신속원형작성에 뒤이어 신중한 면담을 진행하기때문에 게인과 사르센의 구조화체계분석공정은 실제적으로 수행하기 간단하다. 최종자료흐름도를 그림 11-29에 보여 주었다.

자료흐름도(DFD)의 중추적요소는 자료의 저장인 특별식사자료(SPEClAL MEAL DATA)이다(그림 11-29의 중심에서 열린 직사각형으로 표시). 저장용자료는 3개의 원천으로부터 얻어 진다. 즉 비행기안내원(FLlGHT ATTENDANT)에 의하여 조사되는 식사보고서(웃단 왼쪽), 예약자료(RESEVATION DATA)의 저장(웃단 중간), 승객(PASSENGER)의 식사값을 포함하고 있는 조사된 우편엽서(웃단 오른쪽)이다. 자료의 저장 특별식사자료로부터 얻어 지는 특정한 special meal details는 그림 11-29의 중간부아래에서 6개의 둥근 직사각형으로 표시된 공정에 의하여 6개의 보고서를 발생시키는데 리용된다.

하나의 보고서를 발생하기 위하여서는 그 보고서의 시작과 마감날자(*start and end date*)가 그 보고서의 령수자로부터 제공되여야 한다(령수자들은 그림 11-29의 아래에 있는 4개의 2중통들의 행으로 표시된다.).

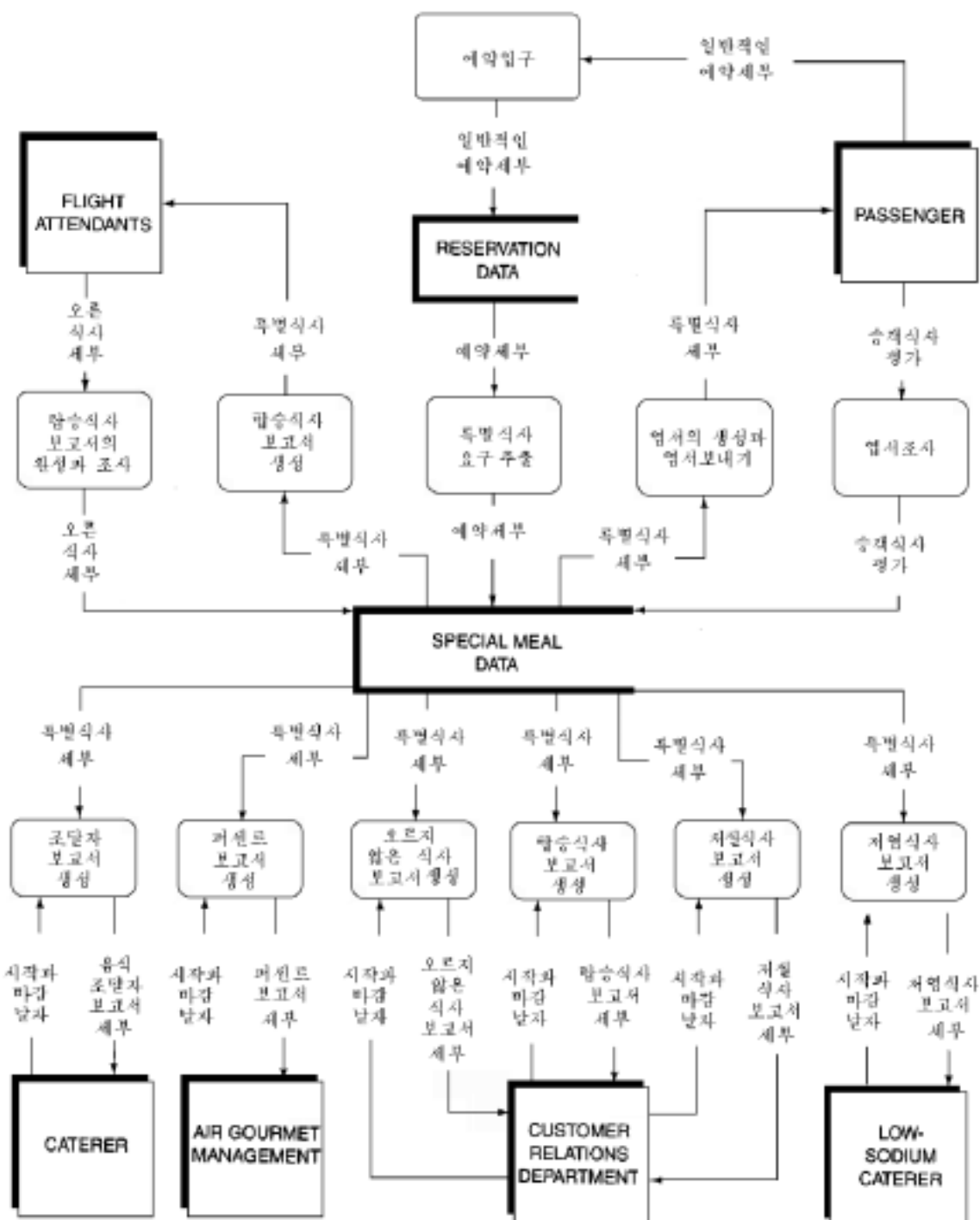
자료흐름도는 자료의 흐름을 고찰하는것으로서 구성되였다. 먼저 매 보고서들을 차례로 고찰하자. 하나의 보고서에 대한 입력은 두개의 부분 즉 special meal details과 start and end date로 구성된다. special meal details는 SPEClAL MEAL DATA의 저장으로부터 생겨 나며 start and end data는 매 보고서의 령수자로부터 제공된다. 그러므로 그림 11-29의 아래단 절반은 매 보고서와 매 보고서의 목적지에 대한 입력자료를 분석함으로써 구성되였다.

SPEClAL MEAL DATA의 원천들이 분석되였다. 즉 자료호출에서 이 부분은 그림 11-29의 웃단절반을 구성하는데 리용되였다. 특별식사자료를 얻을 때 첫단계는 상관된 예약세부(*reservation details*)를 얻는것이다. 이 정보는 예약입력(*enter reservation*)을 처리하기 위하여 PASSENGER에 의하여 제공되며(그림 11-29의 웃단 오른쪽 구석) RESERVATION DATA(그림 11-29의 웃단 절반의 중간에 있는 자료저장)에 저장된다.

그다음 처리특별식사요구추출(*extract special meal requests*)이 DFD에 추가되여 련관된 세부들을 SPEClAL MEAL DATA에 전송한다. 다음으로 탑승자료(*onboard data*)를 고찰하자.

매개의 탑승자료가 FLlGHT ATTENDANT를 위하여 생성되여야 하며 그다음 FLlGHT ATTENDANT는 그 보고서를 완성하고 조사한다. 이 자료흐름은 그림 11-29의 웃단 오른쪽 구석에서 통으로써 모형화되고 화살표로 표식되였다. 앞에서와 마찬가지로 자료흐름도의 이 부분은 련관된 자료흐름, 그것의 원천과 목적지 그리고 그것을 보내는 공정들을 고찰함으로써 구성되였다. 우편엽서의 보내기, 반환, 조사도 류사하게 취급되였다. 매 우편엽서는 SPEClAL MEAL DATA에 있는 자료를 리용하여 생성되며 우편엽서를 쓰고 그것을 돌려 주는 련관된 PASSENGER에게 보내진다. 자료흐름의 이 고리는 그림 11-29의 웃단 오른쪽 구석에서 통으로 모형화되고 화살표로 표식되였다. 또한 일단 이 자료흐름의 원천과 목적지가 식별되고 련관된 공정이 결정되면 이 자료흐름을 DFD에 추가하는것

은 단순하다. 이것으로 그림 11-29의 자료흐름도를 완성한다.

그림 11-29. 항공음식전문회사의 자료흐름도

구조화체계분석의 나머지부분은 부록 5에서 제시된다. 부록 5에서의 자료의 조직과 제시는 의뢰자가 무엇을 작성하려고 하는가를 신속정확히 리해할수 있도록 되여 있다. 그러나 이러한 리해를 가지는데서 중요한 인자는 의뢰자가 요구사항확정단계에서 신속원형을 실험할수 있어야 한다는것이다.

11. 15. 항공음식전문회사 실례연구 : 쏘프트웨어프로젝트관리계획

이제 명세서가 완성되고 비용 및 기한평가를 비롯한 쏘프트웨어프로젝트관리계획이 작성된다고 하자(제9장). 쏘프트웨어개발기업체가 항공음식전문쏘프트웨어제품을 개발하기 위하여 작성한 쏘프트웨어프로젝트관리계획(SPMP)을 포함하고 있다. 이 계획은 1EEE SPMP형식을 만족시킨다(9.5).

11. 16. 명세작성단계에서의 난관

이 장에서 반복되는 화제는 명세서는 의뢰자가 리해하기 충분하게 비형식적이여야 하며 동시에 개발팀이 구성될 제품에 대한 유일한 설명으로서 리용하기 충분하게 형식적이여야 한다는것이다. 명세작성단계에서의 하나의 중요한 난관은 바로 이 모순을 해소하는것이다. 그에 대한 답변은 쉽지 않다. 반대로 이 두 대치되는 목적들사이에는 영구적모순이 놓여 있으며 개발팀은 이러한 진퇴량난을 안전하게 조절하기 위하여 최선을 다해야 한다.

명세작성단계에서의 두번째 난관은 명세서("무엇")와 설계("어떻게")사이의 계선을 혼돈하기 쉽다는것이다. 명세서는 그 제품이 무엇을 하여야 하는가를 서술하여야 한다. 즉 명세서는 그 제품이 어떻게 그것을 하게 되는가를 서술하지 말아야 한다. 실례로 어떤 의뢰자가 일정한 망경로분할계산이 수행될 때마다 0.05s를 넘지 않는 응답시간을 요구한다고 가정하자. 명세서는 이것을 정확하게 서술하여야 하며 그이상은 필요 없다. 특히 명세서는 이러한 응답시간을 달성하기 위하여서는 어느 알고리듬이 리용되여야 하는가를 서술하지 말아야 한다. 즉 하나의 명세서는 모든 제약들을 렬거하여야 하지만 이러한 제약들이 어떻게 달성되는가를 절대로 서술하지 말아야 한다.

이와 같은 잠재적인 함정에 관한 또 하나의 실례는 자료흐름도에서 찾아 볼수 있다(11.3.1). 여기서 둥그런 모서리를 가진 통은 하나의 처리(작용)를 의미하고 있다. 즉 그것은 하나의 모듈을 의미하지 않는다. 11.13에서 설명한바와 같이 자료흐름도(DFD)에서 하나의 처리는 여러개의 각이한 모듈안에 갈라 져 들어 갈수 있으며 또 반대로 여러개의 처리들이 하나의 모듈안에 결합될수도 있다. 중요한 점은 처리로부터 모듈에로의 이와 같은 개선이 명세작성단계가 아니라 설계단계에서 진행되여야 한다는것이다. 명세서는 목적하는 공정의 작용들을 서술하여야 한다. 명세서는 이 작용들이 어떻게 실현되게 되는가를 절대로 명시하지 말아야 하며 매 처리가 할당되는 모듈조차 명시하지 말아야 한다. 설계팀의 과업은 명세서를 총체적으로 연구하고 그 제품의 최량실현을 주게 될

설계를 결정하는것이다. 이에 대하여서는 제13장에서 서술한다.

제품이 전체로써 모듈로 분할되였을 때까지는 특정한 모듈에 대하여 작용을 할당하려는것은 시기상조이다. 그렇게 되면 결과는 거의 틀림없이 준최량적으로 될것이다.

요 약

명세서(11.1)는 비형식적으로(11.2), 준형식적으로(11.3부터 11.5까지) 또는 형식적으로(11.6부터 11.9까지) 표현될수 있다.

이 장의 주요화제는 비형식적기법이 리용하기는 쉽지만 정확하지 않다는것이다. 이것을 한가지 실례로서 보여 준다(11.2.1). 거꾸로 형식적기법들은 강력하지만 숙련하는데서 많은 시간을 투하할것을 필요로 한다(11.10). 한가지 준형식적기법 즉 게인과 사르센의 구조화체계분석이 약간 자세히 서술된다(11.3). 여기서 설명하는 형식적기법들은 유한상태기계(11.6), 페트리망(11.7)과 Z(11.8)이다. 명세서검토에 관한 자료들은 11.12에 서술된다. 그 다음에 명세작성단계에서의 CASE도구들(11.12), 척도들을 설명한다. 그 다음에 항공음식전문회사 실례연구가 취급된다. 즉 그에 대한 구조화체계분석(11.14)과 쏘프트웨어프로젝트관리계획(11.15)이 제시된다. 이 장은 명세작성단계의 난관에 관한 론의로 결속된다(11.16).

보 충

구조화체계분석과 관련한 중요도서들은 문헌 [DeMarco, 1978], [Gane and Sarsen, 1979], [Yordon and Constantine, 1979]이다. 이 책들의 기본착상은 문헌 [Modell, 1996]에서 갱신되였다. 12.5에 제시된 자료지향설계는 다른 부류의 준형식적명세작성기법들과 통합된 설계기법이다. 그에 대한 설명은 문헌 [Jackson, 1975; Warnier, 1976; and Orr, 1981]에서 찾아 볼수 있다. SADT는 문헌 [Ross, 1985]에서, PSL/PSA는 문헌 [Teichroew and Hershey, 1977]에서 론의된다. SREM에 대한 두개의 정보자원은 문헌 [Alford, 1985, and Scheffer, Stone, and Rzepka, 1985]이다.

여섯가지 형식적기법들을 문헌 [Wing, 1990]에서 서술하였다. 형식적기법들에 대한 우수한 론문들은 *IEEE Transactions on Software Engineering* 1990년 11월호, *IEEE Computer, IEEE Software, ACM SIGSOFT Software Engineering Notes*에서 찾아 볼수 있다. 여기서 문헌 [Hall, 1990]은 특별히 흥미가 있다. 문헌 [Bowen and Hinchey, 1995b]은 홀(Hall)의 토론기사에 대한 련속이며 문헌 [Bowen and Hinchey, 1995a]은 형식적기법들의 리용에 관한 지도서로 된다. 형식적기법들에 대한 보충적인 기사들은 *IEEE Transactions on Software Engineering* 1995년 2월호에서 찾아 볼수 있다. 형식적명세작성기법을 산업에 적용한 경우에 얻은 경험들이 문헌 [Larsen, Fitzgerald and Brooks, 1996, and Pfleeger and Hatton, 1997]에서 론의되였다. 형식적방법들에 대한 각이한 견해들은 문헌 [Saiedian et al., 1996]에서 찾아 볼수 있다. 형식적명세작성을 위한 완성모형은 문헌 [Fraser and Vaishnavi, 1997]에서 론의된다.

유한상태기계방법에 대한 초기의 참고문헌은 문헌 [Naur, 1964]인데 이것은 유감스럽게도 튜링기계방법으로 간주되고 있다. 상태도표들은 FSM에 대한 하나의 강력한 확장으로 되는데 이에 대하여서는 문헌 [Harel et al., 1990]에서 서술된다. 상태도표들에 대한 객체지향확장은 문헌 [Coleman, Hayes, and Bear, 1992, and Harel and Gery, 1997]에서 소개되였다.

문헌 [Peterson, 1981]은 페트리망과 그 응용에 대한 훌륭한 소개도서로 된다. 원형작성에서 페트리망의 응용은 문헌 [Bruno and Machetto, 1986]에서 서술되였다. 시간페트리망은 문헌 [Coolahan and Roussopoulos, 1983]에서 서술되였다. Z와 관련하여 문헌 [Diller, 1994]은 훌륭한 소개도서로 된다. 명세작성언어에 대한 완전한 세부를 주는 참고지도서에 관하여서는 문헌 [Spivey, 1992]을 보시오. Z명세서를 리해할 때의 실험결과들을 리용하여 문헌 [Finney, 1996]에서는 Z명세서들이 일부 Z제안자들이 주장하는것처럼 리해하기 쉬운가에 대한 의문을 제기하고 있다.

실시간체계의 명세작성은 *IEEE Transactions on Software Engineering* 1992년 11월호에서 론의된다. 분산체계의 명세작성은 문헌 [Kramer, 1994]에서 론의되였다. 쏘프트웨어 명세작성과 설계에 관한 국제토론회 회보는 명세작성과 관련한 연구착상들에 대한 우수한 자원으로 된다.

문 제*)

11.1. 다음의 제약들은 왜 명세서에 나타나지 말아야 하는가?

ㄱ) 이 쏘프트웨어제품은 우리의 식료품을 서부카나다에 배달하는데서 생기는 수송비용을 본질적으로 감소시켜야 한다.

ㄴ) MRI기억장치는 합리적인 가격으로 설정되여야 한다.

11.2. 다음과 같은 구운 물고기료리법을 고찰하자.

원료:

큰 옥파 1개
랭동 오렌지쥬스 1통
신선하게 짜낸 1개의 레몬쥬스
빵가루 1고뿌
밀가루
우유
중간크기의 서양파 3개

---

*) 문제 11.16(과정안상 목표)와 문제 11.20과 문제 11.21(실례연구)는 11장과 11장의 마지막에서 진행할수 있다.

중간크기의 가지 2개
신선한 물고기 1마리
포일리 흐이쎄(Pouilly Fuissé) 반고뿌
마늘 1개
파르마치즈
닭알 4개

저녁전에 레몬 1개를 즙을 내고 거르어서 그것을 랭동한다. 큰 옥파 1개와 3개의 서양파를 네모나게 자르고 그것들을 후라이판에서 굽는다. 검은 연기가 없어 지기 시작할 때 신선한 오렌지쥬스 2고뿌를 넣는다. 그다음 그것을 잘 뒤섞는다. 레몬을 종이장두께로 썰고 혼합기에 넣는다. 그동안에 버섯에 밀가루를 입히고 그것들을 우유에 담근 다음 빵가루가 있는 종이봉지안에서 그것들을 굴린다. 국남비에 반고뿌의 포일리 흐이쎄를 넣고 가열한다. 170℃에 이르면 사탕가루를 넣고 계속 가열한다. 사탕가루가 캬라멜처럼 되였을 때 버섯을 넣는다. 10min동안 또는 모든 덩어리들이 없어 질 때까지 혼합기로 섞는다. 닭알을 넣는다. 이제 물고기를 집어서 거기에 프롭스(*frobs*)를 부어서 물고기를 죽인다. 물고기껍질을 벗기고 그것을 큰 덩어리로 토막내고 혼합기에 넣는다. 끓이기 시작해서 부글부글 끓어 오르면 뚜껑을 연다. 닭알들은 사전에 5min동안 휘젓개로 잘 휘저어야 한다. 물고기가 만문해 지면 그것을 큰 접시에 담고 파르마치즈를 붓고 4min미만 굽는다.
이 명세서에서 애매성, 생략, 모순들을 결정하시오.

11.3. 의뢰자의 요구를 더 정확히 반영하도록 11.2의 명세서단락을 수정하시오.

11.4. 수학공식을 리용하여 11.2의 명세서단락을 표현하시오. 그것을 문제 11.3에 대한 답변과 비교하시오.

11.5. 은행계산서가 정확한가를 결정하기 위한 제품에 대한 정확한 영어명세서를 작성하시오(문제 8.8).

11.6. 문제 11.5에 대하여 작성한 명세서에 대한 자료흐름도를 그리시오. 작성한 DFD가 자료의 흐름을 간단히 반영하고 있으며 콤퓨터화가 진행되였다는 아무런 가정도 하지 않고 있다는것을 확인하시오.

11.7. 문제 8.7의 자동서고순환체계를 고찰하시오. 서고순환체계에 대한 정확한 명세서를 작성하시오.

11.8. 문제 8.7의 서고순환체계의 조작을 보여 주는 자료흐름도를 그리시오.

11.9. 게인과 사르센의 기법을 리용하여 문제 8.7의 서고순환체계에 대한 명세서를 완성하시오. 거기서 자료는 명시되지 않으며(실제로 매일 장부에 등록되고 삭제되는 책의 총 수) 자기자신이 가정하지만 그것들이 명백하게 지적된다는것을 확인하시오.

11.10. 하나의 류동소수점 2진수는 하나의 선택부호, 한개이상의 비트, 문자 E, 또 하나의 선택부호, 하나이상의 비트로 이루어 진다. 류동소수점 2진수의 실례로 11010E−1010, −100101E11101, +1E0을 들수 있다. 보다 형식적으로 이것을 다음과 같이 표현할수 있다.

<floating-point binary>:: = [<sign>]<bitstring>E[<sign>]<bitstring>

<sign> :: = + | -

<bitstring> :: = <bit>[<bitstring>]

<bit> :: = 0 | 1

(여기서 […]은 선택항목을 의미하며 a|b는 a 또는 b를 의미한다.).
입력을 문자렬로 취하며 문자렬이 타당한 류동소수점 2진수로 구성되여 있는가를 결정하는 유한상태기계를 명시하시오.

11.11. 유한상태기계방법을 리용하여 문제 8.7의 서고순환체계를 명시하시오.

11.12. 문제 11.11에 대한 당신의 답이 서고순환체계를 위한 차림표구동제품을 설계하고 실현하는데 어떻게 리용될수 있는가를 보여 주시오(문제 8.7).

11.13. 페트리망을 리용하여 한권의 책이 문제 8.7의 서고를 순환하는것을 명시하시오. 명세서에 조작 H, C 및 R를 포함시키시오.

11.14. 당신이 도서관체계를 전문으로 콤퓨터화할 어떤 큰 회사에 근무하는 쏘프트웨어기사라고 하자. 관리자는 당신에게 문제 8.7의 완전한 서고순환체계를 Z로 명시할것을 요구하고 있다. 당신의 답변은 무엇인가?

11.15. (과정안상 목표) 교원이 규정한 기법을 리용하여 부록 1에 서술된 브로드랜즈지역 아동병원제품에 대한 명세서를 작성하시오.

11.16. (과정안상 목표) 부록 1에 서술된 브로즈랜즈지역 아동병원제품을 위한 쏘프트웨어프로젝트관리계획을 작성하시오.

11.17. (실례연구) 유한상태기계방법을 리용하여 항공음식전문회사제품에 대한 요구사항을 작성하시오.

11.18. (실례연구) 페트리망을 리용하여 항공음식전문회사제품에서 한개의 특별식사가 통과하는 상태들을 명시하시오.

11.19. (실례연구) 11.8의 Z구성을 리용하여 항공음식전문회사제품의 한개 부분을 명시하시오.

11.20. (실례연구) 11.15의 쏘프트웨어프로젝트관리계획은 세명의 쏘프트웨어기사로 구성된 소규모의 쏘프트웨어기업체를 위한것이다. 1,000명이상의 쏘프트웨어기사를 가진 중규모의 쏘프트웨어기업체에 적합하도록 이 계획을 변경하시오.

11.21. (실례연구) 만일 항공음식전문회사제품이 8주동안에 완성되여야 한다면 11.15의 쏘프트웨어프로젝트관리계획은 무슨 방식으로 변경되여야 하는가?

11.22. (쏘프트웨어공학독본) 교원이 [Bowen and Hinchey, 1995b]의 복제본을 배포할것이다. 당신은 보웬과 힌체이의 지위에 동의하는가 또는 당신은 《일곱가지 희랍신화》의 일부가 적어도 부분적으로 사실이라고 생각하는가? 당신의 대답을 정당화하시오.

참 고 문 헌

[Abrial, 1980] J.-R. ABRIAL, "The Specification Language Z: Syntax and Semantics," Oxford University Computing Laboratory, Programming Research Group, Oxford, U.K., April 1980.

[Alford, 1985] M. ALFORD, "SREM at the Age of Eight; The Distributed Computing Design System," *IEEE Computer* **18** (April 1985), pp. 36–46.

[Balzer, 1985] R. BALZER, "A 15 Year Perspective on Automatic Programming," *IEEE Transactions on Software Engineering* **SE-11** (November 1985), pp. 1257–68.

[Banks, Carson, Nelson, and Nichol, 2000] J. BANKS, J. S. CARSON, B. L. NELSON, AND D. M. NICHOL, *Discrete-Event System Simulation,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1995.

[Bowen and Hinchey, 1995a] J. P. BOWEN AND M. G. HINCHEY, "Ten Commandments of Formal Methods," *IEEE Computer* **28** (April 1995), pp. 56–63.

[Bowen and Hinchey, 1995b] J. P. BOWEN AND M. G. HINCHEY, "Seven More Myths of Formal Methods," *IEEE Software* **12** (July 1995), pp. 34–41.

[Brady, 1977] J. M. BRADY, *The Theory of Computer Science,* Chapman and Hall, London, 1977.

[Bruno and Marchetto, 1986] G. BRUNO AND G. MARCHETTO, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Transactions on Software Engineering* **SE-12** (February 1986), pp. 346–57.

[Chen, 1976] P. CHEN, "The Entity-Relationship Model—Towards a Unified View of Data," *ACM Transactions on Database Systems* **1** (March 1976), pp. 9–36.

[Coleman, Hayes, and Bear, 1992] D. COLEMAN, F. HAYES, AND S. BEAR, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," *IEEE Transactions on Software Engineering* **18** (January 1992), pp. 9–18.

[Coolahan and Roussopoulos, 1983] J. E. COOLAHAN, JR., AND N. ROUSSOPOULOS, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," *IEEE Transactions on Software Engineering* **SE-9** (September 1983), pp. 603–16.

[Dart, Ellison, Feiler, and Habermann, 1987] S. A. DART, R. J. ELLISON, P. H. FEILER, AND A. N. HABERMANN, "Software Development Environments," *IEEE Computer* **20** (November 1987), pp. 18–28.

[Delisle and Garlan, 1990] N. DELISLE AND D. GARLAN, "A Formal Description of an Oscilloscope," *IEEE Software* **7** (September 1990), pp. 29–36.

[Delisle and Schwartz, 1987] N. DELISLE AND M. SCHWARTZ, "A Programming Environment for CSP," Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, *ACM SIGPLAN Notices* **22** (January 1987), pp. 34–41.

[DeMarco, 1978] T. DEMARCO, *Structured Analysis and System Specification,* Yourdon Press, New York, 1978.

[Diller, 1994] A. DILLER, *Z: An Introduction to Formal Methods,* 2nd ed., John Wiley and Sons, Chichester, U.K., 1994.

[Doolan, 1992] E. P. DOOLAN, "Experience with Fagan's Inspection Method," *Software—Practice and Experience* **22** (February 1992), pp. 173–82.

[Fagan, 1976] M. E. FAGAN, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15** (No. 3, 1976), pp. 182–211.

[Finney, 1996] K. FINNEY, "Mathematical Notation in Formal Specification: Too Difficult for the Masses?" *IEEE Transactions on Software Engineering* **22** (1996), pp. 158–59.

[Fraser and Vaishnavi, 1997] M. D. FRASER AND V. K. VAISHNAVI, "A Formal Specifications Maturity Model," *Communications of the ACM* **40** (May 1997), pp. 95–103.

[Gane and Sarsen, 1979] C. GANE AND T. SARSEN, *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, Englewood Cliffs, NJ, 1979.
[Ghezzi and Mandrioli, 1987] C. GHEZZI AND D. MANDRIOLI, "On Eclecticism in Specifications: A Case Study Centered around Petri Nets," *Proceedings of the Fourth International Workshop on Software Specification and Design*, Monterey, CA, 1987, pp. 216–24.
[Goodenough and Gerhart, 1975] J. B. GOODENOUGH AND S. L. GERHART, "Toward a Theory of Test Data Selection," *Proceedings of the Third International Conference on Reliable Software*, Los Angeles, 1975, pp. 493–510. Also published in *IEEE Transactions on Software Engineering* **SE-1** (June 1975), pp. 156–73. Revised version: J. B. Goodenough, and S. L. Gerhart, "Toward a Theory of Test Data Selection: Data Selection Criteria," in *Current Trends in Programming Methodology*, Volume 2, R. T. Yeh (Editor), Prentice Hall, Englewood Cliffs, NJ, 1977, pp. 44–79.
[Gordon, 1979] M. J. C. GORDON, *The Denotational Description of Programming Languages: An Introduction*, Springer-Verlag, New York, 1979.
[Guha, Lang, and Bassiouni, 1987] R. K. GUHA, S. D. LANG, AND M. BASSIOUNI, "Software Specification and Design Using Petri Nets," *Proceedings of the Fourth International Workshop on Software Specification and Design*, Monterey, CA, April 1987, pp. 225–30.
[Hall, 1990] A. HALL, "Seven Myths of Formal Methods," *IEEE Software* **7** (September 1990), pp. 11–19.
[Harel and Gery, 1997] D. HAREL AND E. GERY, "Executable Object Modeling with Statecharts," *IEEE Computer* **30** (July 1997), pp. 31–42.
[Harel et al., 1990] D. HAREL, H. LACHOVER, A. NAAMAD, A. PNUELI, M. POLITI, R. SHERMAN, A. SHTULL-TRAURING, AND M. TRAKHTENBROT, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering* **16** (April 1990), pp. 403–14.
[Hoare, 1985] C. A. R. HOARE, *Communicating Sequential Processes*, Prentice Hall International, Englewood Cliffs, NJ, 1985.
[IWSSD, 1986] Call for Papers, Fourth International Workshop on Software Specification and Design, *ACM SIGSOFT Software Engineering Notes* **11** (April 1986), pp. 94–96.
[Jackson, 1975] M. A. JACKSON, *Principles of Program Design*, Academic Press, New York, 1975.
[Jones, 1986b] C. B. JONES, *Systematic Software Development Using VDM*, Prentice Hall, Englewood Cliffs, NJ, 1986.
[Kampen, 1987] G. R. KAMPEN, "An Eclectic Approach to Specification," *Proceedings of the Fourth International Workshop on Software Specification and Design*, Monterey, CA, April 1987, pp. 178–82.
[Kleinrock and Gail, 1996] L. KLEINROCK AND R. GAIL, *Queuing Systems: Problems and Solutions*, John Wiley and Sons, New York, 1996.
[Knuth, 1968] D. E. KNUTH, *The Art of Computer Programming*, Volume 1, *Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
[Kramer, 1994] J. KRAMER, "Distributed Software Engineering," *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, pp. 253–63.
[Larsen, Fitzgerald, and Brookes, 1996] P. G. LARSEN, J. FITZGERALD, AND T. BROOKES, "Applying Formal Specification in Industry," *IEEE Software* **13** (May 1996), pp. 48–56.
[Leavenworth, 1970] B. LEAVENWORTH, Review #19420, *Computing Reviews* **11** (July 1970), pp. 396–97.

[London, 1971] R. L. LONDON, "Software Reliability through Proving Programs Correct," *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing,* March 1971.

[Luckham and von Henke, 1985] D. C. LUCKHAM AND F. W. VON HENKE, "An Overview of Anna, a Specification Language for Ada," *IEEE Software* **2** (March 1985), pp. 9–22.

[Meyer, 1985] B. MEYER, "On Formalism in Specifications," *IEEE Software* **2** (January 1985), pp. 6–26.

[Modell, 1996] M. E. MODELL, *A Professional's Guide to Systems Analysis,* 2nd ed., McGraw-Hill, 1996.

[Naur, 1964] P. NAUR, "The Design of the GIER ALGOL Compiler," in: *Annual Review in Automatic Programming,* Volume 4, Pergamon Press, Oxford, U.K., 1964, pp. 49–85.

[Naur, 1969] P. NAUR, "Programming by Action Clusters," *BIT* **9** (No. 3, 1969), pp. 250–58.

[Nix and Collins, 1988] C. J. NIX AND B. P. COLLINS, "The Use of Software Engineering, Including the Z Notation, in the Development of CICS," *Quality Assurance* **14** (September 1988), pp. 103–10.

[Oest, 1986] O. N. OEST, "VDM from Research to Practice," *Proceedings of the IFIP Congress, Information Processing '86,* 1986, pp. 527–33.

[Orr, 1981] K. ORR, *Structured Requirements Definition,* Ken Orr and Associates, Topeka, KS, 1981.

[Peterson, 1981] J. L. PETERSON, *Petri Net Theory and the Modeling of Systems,* Prentice Hall, Englewood Cliffs, NJ, 1981.

[Petri, 1962] C. A. PETRI, "Kommunikation mit Automaten," Ph.D. Dissertation, University of Bonn, Germany, 1962. [In German.]

[Pfleeger and Hatton, 1997] S. L. PFLEEGER AND L. HATTON, "Investigating the Influence of Formal Methods," *IEEE Computer* **30** (February 1997), pp. 33–43.

[Pollack, Hicks, and Harrison, 1971] S. L. POLLACK, H. T. HICKS, JR., AND W. J. HARRISON, *Decision Tables: Theory and Practice,* Wiley-Interscience, New York, 1971.

[Ross, 1985] D. T. ROSS, "Applications and Extensions of SADT," *IEEE Computer* **18** (April 1985), pp. 25–34.

[Saiedian et. al., 1996] H. SAIDEIAN, J. P. BOWEN, R. W. BUTLER, D. L. DILL, R. L. GLASS, D. GRIES, A. HALL, M. G. HINCHEY, C. M. HOLLOWAY, D. JACKSON, C. B. JONES, M. J. LUTZ, D. L. PARNAS, J. RUSHBY, J. WING, AND P. ZAVE, "An Invitation to Formal Methods," *IEEE Computer* **29** (April 1996), pp. 16–30.

[Scheffer, Stone, and Rzepka, 1985] P. A. SCHEFFER, A. H. STONE, III, AND W. E. RZEPKA, "A Case Study of SREM," *IEEE Computer* **18** (April 1985), pp. 47–54.

[Schwartz and Delisle, 1987] M. D. SCHWARTZ AND N. M. DELISLE, "Specifying a Lift Control System with CSP," *Proceedings of the Fourth International Workshop on Software Specification and Design,* Monterey, CA, April 1987, pp. 21–27.

[Silberschatz and Galvin, 1998] A. SILBERSCHATZ AND P. B. GALVIN, *Operating System Concepts,* 5th ed., Addison-Wesley, Reading, MA, 1998.

[Spivey, 1990] J. M. SPIVEY, "Specifying a Real-Time Kernel," *IEEE Software* **7** (September 1990), pp. 21–28.

[Spivey, 1992] J. M. SPIVEY, *The Z Notation: A Reference Manual,* Prentice Hall, New York, 1992.

[Teichroew and Hershey, 1977] D. TEICHROEW AND E. A. HERSHEY, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering* **SE-3** (January 1977), pp. 41–48.

[Warnier, 1976] J. D. WARNIER, *Logical Construction of Programs*, Van Nostrand Reinhold, New York, 1976

[Wing, 1990] J. WING, "A Specifier's Introduction to Formal Methods," *IEEE Computer* **23** (September 1990), pp. 8–24.

[Woodcock, 1989] J. WOODCOCK, "Calculating Properties of Z Specifications," *ACM SIGSOFT Software Engineering Notes* **14** (July 1989), pp. 43–54.

[Yourdon and Constantine, 1979] E. YOURDON AND L. L. CONSTANTINE, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice Hall, Englewood Cliffs, NJ, 1979.

제１２장. 객체지향분석단계

명세작성단계를 때로는 분석단계(*analysis phase*)라고도 부른다. 이것은 고전적명세작성단계에 대응하는 객체지향방법을 객체지향분석단계(*object-oriented analysis phase*)라고 부르기때문이다. 이 장은 제11장에 대응하는 객체지향방법으로 간주할수 있다.

１２．１．객체지향분석

객체지향분석(*Object-oriented analysis*; OOA)은 객체지향파라다임에 대한 준형식적명세작성기법이다. 제11장에서는 여러가지 서로 다른 기법들이 구조화체계분석에 리용되며 그것들은 모두 본질적으로 등가이라는것을 지적하였다. 이와 류사하게 현재 60개이상의 각이한 기법들이 OOA에 리용되고 있는데 새로운 기법들은 하나의 규칙에 토대하여 제시된다. 또한 이 모든 기법들은 대부분 등가이다. 이 장의 《보충》부분에는 각이한 기법들은 물론 그 기법들사이의 비교를 소개한 참고서들이 있다. 객체지향분석은 하나의 준형식적기법이기때문에 OOA에서의 매개 기법의 본질적인 부분은 그 기법과 관련된 도형표시법이다. 그러므로 어떤 특정한 기법의 리용방법을 배우는것은 그 기법에서의 련관된 도형표시법을 배운다는것을 의미한다.

이것은 통합모형화언어(*Unified Modeling Language*; UML)의 발표와 함께 변화되였으며 [Rumbangh, Jacobson, and Booch, 1999]하나의 공통적인 표식법이 매 OOA기법으로 리용되도록 설계되였다. 이 책에서 UML은 객체지향분석과 객체지향설계를 모두 표현하기 위하여 리용된다.

UML에 대한 보충적인 정보는 다음의 《알고 싶은 문제》를 보시오.

객체지향분석은 세단계로 이루어 진다(그것을 아래의 란에 종합하였다.).

1. 유스-케스모형화(Use-case modelling) 이것은 각이한 결과들이 제품에 의해 어떻게 계산되는가를 결정한다(순서는 고려하지 않는다.). 이 정보를 하나의 유스-케스도(*use-case diagram*)와 그와 련관된 대본들의 형식으로 제시한다(10.12에서 서술한 대본은 유스-케스의 한가지 실례이다.). 이 단계를 때로는 기능적모형화(*functional modeling*)로 부르는데 이것은 대체로 작용지향적이다.

2. 클라스모형화(Class modeling) 클라스들과 그것들의 속성을 결정한다.

그다음 클라스들사이의 호상관계와 호상작용을 결정한다. 이 정보를 클라스도(*class diagram*)라고 부르는 실체-관련도와 약간 류사한 도식의 형식으로 제시한다(일부 저자들은 이 도식을 클라스모형(*class model*)이라고 부른다.).

3. 동적모형화(Dynamic modeling) 매 클라스 또는 부분클라스들에 의하여 또는 그에 대하여 수행되는 작용을 결정한다. 이 정보를 상태도(*state diagram*)라고 부르는 유한상태기계(11.6)와 약간 류사한 형식으로 제시한다. 이 단계는 작용지향적이다.

알고 싶은 문제

최근까지 가장 보편적인 객체지향분석과 설계방법론은 객체모형화기술 (OMT)[Rumbaugh et al., 1991]과 Booch의 기법[Booch, 1994]이였다.

OMT는 짐 룸바우(Jim Rumbaugh)에 의해 개발되였으며 그의 연구팀은 뉴욕의 스체넥타디(Schenetady)에 있는 제네랄 엘렉트리니크연구개발쎈터에 있다. 한편 글레디 브츠(Grady Booch)는 캘리포니아의 싼타 클라라(Santa Clara)에 있는 래숀널주식회사(Rational, Inc.)에서 자기의 방법론을 개발하였다.

이미 설명한바와 같이 모든 객체지향분석기법들은 본질적으로 등가이다. 객체지향설계기법들도 역시 대체로 서로 등가이며 따라서 OMT와 브츠의 방법론사이의 차이는 작다. 그러나 이 두 진영의 지지자들사이에는 언제나 승벽내기가 존재하였다.

이 상황은 1994년 10월 룸바우가 래숀널주식회사에서 브츠와 합류함으로써 변화되였다. 이 두명의 방법론자들은 즉시 자기들의 방법을 결합한데 기초하여 그들이 통합방법론(*Unified Methodology*)이라고 명명한 방법을 개발하기 시작하였다. 그들의 첫 연구결과가 발표되였을 때 그들이 하나의 방법론이 아니라 순전히 객체지향쏘프트웨어제품을 표현하기 위한 한가지 표시법을 개발하였다는것이 지적되였다. 통합방법론이라는 이름은 인차 통합모형화언어(*Unified Modeling Language*)로 바뀌였다.

1995년에 객체지향쏘프트웨어공학의 선구자인 이와 재콥숀(Ivar Jacobson)이 래숀널주식회사에서 그들과 합류하였다. 재콥숀은 1967년부터 시작하여 스웨리예에서 객체지향쏘프트웨어공학이라는 자기의 방법론[Jacobson, Christerson, Jonson, and Overgaard, 1992]을 개발하였다. UML1.0판은 1997년에 발표되였다. 그것은 룸바우, 재콥숀, 브츠의 공동연구를 반영하고 있으며 때로 그들은 종합적으로 세 동료(*three amigos*)로 불리운다. UML은 지금 하나의 국제적인 표준이며 그것은 객체관리그룹(*Object Management Group*)의 감독하에 개정되고 확장되고 있으며 전 세계적인 회사재단이 객체지향파라다임분야에서 활약하고 있다.

다음으로 재콥숀, 브츠, 룸바우는 자기들의 이런 개별적방법들을 결합(통합화)한 쏘프트웨어개발방법들은 공동으로 창시하였다. 그들이 창시한 통합쏘프트웨어개발공정(*Unified Software Development Process*)[Jacobson, Booch and Rumbaugh, 1999]을 합리적인 통합공정(*Rational Unified Precess*)이라고 형식적으로 부르는데 이것은 첫 UML에 기초한 쏘프트웨어 개발방법론이다. 또 하나의 중요한 UML에 기초한 방법론은 Catalysis이다[D'Souza and Wills, 1999]. UML은 미래의 쏘프트웨어개발방법들도 UML에 기초할 정도로 그렇게 광범하게 리용되고 깊이 침투하였다.

객체지향분석을 진행하는 방법

다음의 세단계를 반복한다 :

- 유스-케스모형화(*Use-case modeling*).
  각이한 결과들이 제품에 의하여 어떻게 계산되는가를 결정한다.
- 클라스모형화(*Class modeling*)
  클라스, 클라스의 속성, 클라스들사이의 호상관계를 결정한다.
- 동적모형화(*Dynamic modeling*)
  매 클라스 또는 부분클라스에 의하여 또는 그에 대하여 수행되는 작용들을 결정한다.

실천적으로 이 세 단계들은 순수 순차적으로 수행되지는 않는다. 한 도식에서의 변화는 다음 두 도식에서의 대응하는 수정을 유발시킨다. 결국 OOA의 세단계는 병렬로 효과적으로 수행된다. 이것은 객체지향파라다임에서 자료(2단계)도, 작용(1단계와 3단계)들도 다른것보다 앞설수 없기때문에 타당하다.

OOA는 확실히 앞장에서 고찰한 두개의 구조화된 명세작성기법 즉 실체-관련모형화(ERM)와 유한상태기계(FSM)의 결합으로서 간주하지 말아야 한다. 반대로 OOA는 3개의 모형화단계들의 결과를 명시하기 위하여 도식들을 리용하는 하나의 모형화기법이다. 이 정보를 현시하는 새로운 방법들을 총체적으로 창안하는것 대신에 광범히 리용되고 있는 구조화된 기법들에 기초한 표시법을 받아 들이는것이 타당하다.

이것을 고찰하기 위한 또 한가지 방법은 OOA의 목적이 기타 모든 명세작성기법들과 마찬가지로 구성될 목표제품을 명시하는것이라는것을 옳바로 인식하는것이다. 제품의 두가지 위험한 측면은 그 제품의 자료와 작용들이다. OOA는 각이한 모형화기법들을 리용하여 자료, 작용들 그리고 그것들사이의 호상작용을 리해한다. 분석과정에서 그 제품에 관하여 얻어 지는 지식은 각이한 방식으로 표현되며 그 매개는 목적하는 제품의 각이한 측면들을 반영하고 있다. 선도들은 모형화되는 체계에 대한 보다 철저한 리해가 이루어짐에 따라서 련속적으로 갱신된다. OOA단계의 마감에서 통합된 조사가 제품에 대한 총체적인 리해를 제공하게 되는데 이것은 오직 한가지 모형화기법이 도입되였다면 달성하기 어려울수 있다.

12. 2. 승강기문제: 객체지향분석

OOA의 단계들을 한가지 실례로서 서술한다. 승강기문제는 제11장에서 처음으로 설명하였다. 참고하기 쉽도록 하기 위하여 여기서 그 문제를 다시 반복한다.

이 제품은 m층으로 된 어떤 건물에 있는 n개의 승강기를 조종하기 위하여 설치되게 된다. 이 문제는 다음과 같은 제약에 따라 m층사이에서 n개의 승강기를 움직이는데 필요한 론리와 관련된 문제이다.

1. 매 승강기는 m개의 단추모임을 가지고 있는데 매 층에 하나의 단추가 대응한다. 단추들은 눌러 질 때 조명이 켜지며 승강기가 대응하는 층을 찾아 가게 된다. 승강기가 대응하는 층을 찾아 갔을 때 단추의 조명은 꺼진다.
2. 1층과 마지막층을 제외하고 매층은 두개의 단추를 가진다. 하나의 단추는 승강기가 올라 갈것을 요청하며 다른 하나는 승강기가 내려 갈것을 요청하는 단추이다. 이 단추들을 누르면 조명이 켜진다. 승강기가 해당한 층을 찾아 간 다음 희망하는 방향으로 움직일 때 조명은 꺼진다.
3. 승강기에 아무런 요청도 없을 때 승강기는 문을 닫은채로 현재의 층에 머물러 있다.

OOA에서 첫단계는 유스케스(*use case*)를 모형화하는것이다.

1 2. 3. 유스-케스모형화(Use-Case Modelling)

유스케스는 구성될 제품의 기능성을 서술한다. 그것은 전체적인 기능성에 대한 일반적인 서술을 제공하여 준다. 그러면 대본들은 마치 객체들이 어떤 클라스의 실례들인것처럼 유스케스의 특정한 실례들로 된다. 하나의 유스케스는 쏘프트웨어제품의 클라스들과 그 제품의 사용자들사이의 전체적인 호상작용과 관련된다. 하나의 대본은 특정한 객체들과 사용자들사이의 하나의 특수한 호상작용들의 모임이다.

승강기문제에 대한 유스케스의 UML표현을 그림 12-1에 보여 주었다. 사용자들과 클라스들사이에 가능한 유일한 호상작용은 사용자가 하나의 승강기를 호출하려고 승강기단추를 누르거나 사용자가 승강기를 어떤 특정한 층에 세우려고 층단추를 누르는것이다. 전체적인 기능에 관한 이러한 일반적서술내에서 많은 서로 다른 대본들을 뽑아 낼수 있는데 그 매 대본들은 호상작용들의 하나의 특정한 모임을 표현하고 있다. 실례로 그림 12-2는 하나의 표준대본*)을 묘사하고 있다. 즉 우리가 승강기가 어떻게 리용되여야 하는가를 리해하는 방법에 대응하는 사용자들과 승강기들사이의 호상작용들의 모임을 서술하고 있다.

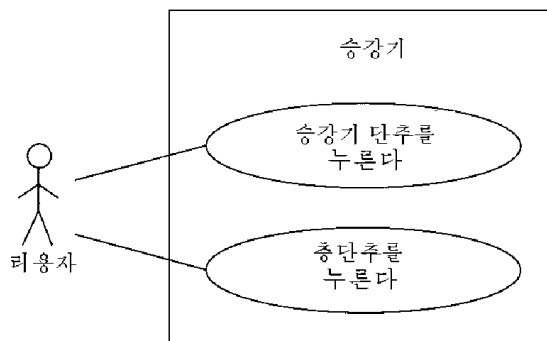



그림 12-1. 승강기문제의 유스-케스도

그림 12-2는 승강기(보다 정확하게는 승강기단추들과 층단추들)와 호상작용하는 각이한 사용자들을 주의 깊게 관찰한후 구성되였다. 그림의 15개의 사건들은 사용자 A와 승강기체계의 단추들사이의 두개의 호상작용(사건 1과 사건 7)과 승강기체계의 요소들이 취하는 작용들(사건 2부터 사건 6까지와 사건 8부터 사건 15까지)을 자세히 서술하고 있다.

두개의 항목 즉 《사용자 A가 승강기에 오른다.》와 《사용자 A가 승강기에서 내린다.》는 번호를 붙이지 않았다. 이와 같은 항목들은 본질상 설명에 해당한다. 즉 사용

---

*) UML은 대본을 표현하기 위한 두가지 류형의 선도 즉 순차도(*sequence diagram*)와 협동도(*collaboration diagram*)를 제공한다. 그러나 이 선도들은 객체지향분석보다 객체지향설계에서 더 유용하다. 그러므로 이 선도들에 대한 론의는 13.7에서 진행한다.

자 A는 승강기에 오르거나 내릴 때 승강기의 요소들과 아무런 호상작용도 하지 않는다. 반대로 그림 12-3은 하나의 례외대본이다. 이 대본은 사용자가 3층에서 상승단추를 눌렀지만 사실은 1층으로 내려 가려고 할 때 일어 나는 정황을 묘사하고 있다. 이 대본 역시 승강기를 타는 많은 사용자들의 행동을 관찰한 결과로 구성되였다. 즉 승강기를 전혀 리용해 본적이 없는 누군가가 사용자들이 때로 단추를 잘못 누른다는것을 리해하지 못한것 같다. 승강기만큼 파악되지 못한 령역에서는 대본들이 요구서로부터 얻어 지게 된다. 이에 대하여서는 12.8에서 서술한다.

1. 사용자 A는 3층에서 상승층단추를 눌러 승강기를 요청한다.
   사용자 A는 7층으로 가려고 한다.
2. 상승층단추가 켜진다.
3. 승강기가 3층에 도착한다. 승강기에는 사용자 B가 타고 있는데 그는 1층에서 승강기에 올라 9층 승강기단추를 눌렀다.
4. 상승층단추가 꺼진다.
5. 승강기문이 열린다.
6. 시간계수기가 동작한다.
   사용자 A가 승강기에 오른다.
7. 사용자 A는 7층 승강기단추를 누른다.
8. 7층 승강기단추가 켜진다.
9. 설정시간이 지나면 승강기문이 닫긴다.
10. 승강기가 7층으로 움직인다.
11. 7층승강기단추가 꺼진다.
12. 사용자 A가 승강기에서 내리도록 승강기문이 열린다.
13. 시간계수기가 동작한다.
    사용자 A는 승강기에서 내린다.
14. 설정시간이 지나면 승강기문이 닫긴다.
15. 승강기는 사용자 B를 태우고 9층으로 간다.

그림 12-2. 표준대본의 첫 반복

그림 12-2와 12-3의 대본들은 기타 무수히 많은 대본들과 함께 그림 12-1에 보여 준 유스케스의 특정한 실례들이다. 대본들은 OOA팀이 모형화하려는 체계의 거동을 종합적으로 통찰할수 있도록 충분히 연구되여야 한다. 이 정보는 다음단계 즉 클라스모형화에서 클라스들(객체들)을 결정하기 위하여 리용된다. 이것은 또한 객체지향설계에서 리용된다(13.6).

1. 사용자 A가 3층에서 상승층단추를 눌러 승강기를 요청한다.
   사용자 A는 1층으로 가려고 한다.
2. 상승층단추가 켜진다.
3. 승강기가 3층에 도착한다. 승강기에는 사용자 B가 타고 있는데 그는 1층에서 승강기에 올라 9층 승강기단추를 눌렀다.
4. 상승층 단추가 꺼진다.
5. 승강기문이 닫긴다.
6. 시간계수기가 동작한다.
   사용자 A가 승강기에 오른다.
7. 사용자 A는 1층승강기단추를 누른다.
8. 1층승강기단추가 켜진다.
9. 설정시간이 지나면 승강기문이 닫긴다.
10. 승강기는 9층으로 움직인다.
11. 9층승강기단추가 꺼진다.
12. 사용자 B가 승강기에서 내리도록 승강기문이 열린다.
13. 시간계수기가 동작한다.
    사용자 B는 승강기에서 내린다.
14. 설정시간이 지나면 승강기문이 닫긴다.
15. 승강기는 사용자 A를 태우고 1층으로 간다.

그림 12-3. 례외대본

1 2. 4. 클라스모형화

이 단계에서 클라스들과 그 속성들이 실체-관련도를 리용하여 추출되고 표현된다. 이 단계에서는 클라스의 속성들만 결정되고 방법들은 결정되지 않는다. 방법들은 객체지향설계에서 클라스들에 할당된다.

전체적인 객체지향파라다임의 한가지 특징은 각이한 단계들을 수행하기가 쉽지 않다는것이다. 다행히도 객체를 리용하는데서 얻게 되는 리득이 이러한 노력을 헛되게 하지 않는다. 그러므로 클라스모형화의 맨 첫단계 즉 클라스들과 그것의 속성추출에서 첫번에 옳바른것을 얻기가 일반적으로 어렵다는데 대하여 놀라지 말아야 한다.

클라스들을 결정하는 한가지 방법은 그것들을 유스케스들로부터 연역해 내는것이다. 즉 개발자들은 표준 및 례외대본들을 포함해서 모든 대본들을 주의 깊게 연구하고 유스케스들에서 역할을 노는 요소들을 식별해 낸다. 바로 그림 12-2와 12-3의 대본들에서 후보클라스들은 승강기단추, 층단추, 승강기, 문 그리고 시간계수기들이다. 우리가 보게 되는바와 같이 이 후보클라스들은 클라스모형화기간에 추출되는 실제적인 클라스들에 가깝다. 그러나 일반적인 대본들은 매우 많으며 결과적으로 많은 잠재적클라스들이 존재하게

된다. 경험 없는 개발자는 대본들로부터 너무 많은 후보클라스들을 추론하려고 할수도 있다. 이것은 새로운 클라스를 추가하는것이 포함되지 말아야 하는 후보클라스를 제거하는 것보다 더 쉽기때문에 클라스모형화에 해로운 영향을 줄것이다.

클라스들을 결정하기 위한 또 한가지 방법은 CRC카드(12.4.2)인데 이 방법은 개발자들이 령역전문지식을 가지고 있을 때 효과적이다. 그러나 만일 전문가들이 응용령역에서 경험이 적거나 전혀 없다면 다음절에서 설명한다. 명사추출(*noun extraction*)을 리용하는것이 현명한 처사이다.

12. 4. 1. 명사추출

령역전문지식이 없는 개발자들에게 있어서 한가지 좋은 방법은 다음의 세단계의 과정을 리용하여 후보클라스들을 추출하고 그다음 그것들을 세련하는것이다.

1단계. 간결한 문제정의 제품을 될수록 간단하고 간결하게, 좋기는 하나의 문장으로 정의한다. 승강기문제에서 이를 위한 한가지 방법은 다음과 같다.

> 승강기와 층에 있는 단추들은 m층 건물의 n개의 승강기의 움직임을 조종한다.

2단계. 비형식적전략 문제해결에서 비형식적전략으로 대처하기 위하여서는 제약들이 고려되여야 한다. 승강기문제에 대한 세가지 제약들이 12.2에 제시되였다. 이제 비형식적전략은 오히려 단순단락으로 표시될수 있다. 승강기문제에 대한 한가지 가능한 서술단락은 다음과 같다.

> 승강기와 층에 있는 단추들은 m층건물의 n개의 승강기의 움직임을 조종한다. 단추들은 승강기를 어떤 특정한 층에 세울것을 요청하여 눌렀을 때 조명이 켜진다. 그 요청이 만족되였을 때 조명은 꺼진다. 승강기에 아무런 요청도 없을 때 승강기는 문을 닫은채로 현재의 층에 머물러 있다.

3단계. 전략의 형식화 비형식적전략에서 명사들을 식별(문제범위밖에 놓이는것들을 배제하면서)하고 그다음 이 명사들을 후보클라스로서 리용한다. 이제 비형식적전략을 다시 생성하는데 이번에는 식별된 명사들이 다른 서체로서 인쇄된다.

> **승강기**와 **층**에 있는 **단추**들은 m**층 건물**의 n개의 **승강기**의 **움직임**을 조종한다. **단추**들은 **승강기**를 어떤 특정한 **층**에 세울것을 요청하며 눌렀을 때 **조명**이 켜진다. 그 요청이 만족되였을 때 **조명**은 꺼진다. **승강기**에 아무런 요청도 없을 때 **승강기**는 **문**을 닫은채로 현재의 **층**에 머물러 있다.

여기에는 8개의 서로 다른 명사 즉 단추, 승강기, 층, 움직임, 건물, 조명, 요청, 문이 있다. 이 명사들가운데서 3개 즉 층, 건물, 문은 문제범위밖에 있기때문에 무시될수 있다. 나머지 3개의 명사 즉 움직임, 조명, 요청 등은 추상명사들이다. 즉 그것들은 《물리적존재를 가지지 않는 사상 또는 질로 볼수 있다》(*World Book Encyclopedia*, 2000). 한가지 유용한 경험은 추상명사들은 좀처럼 클라스에 대응시킬수 없다는것이다. 그대신에 그것들은 자주 클라스의 속성으로 된다. 실례로 《조명》은 단추의 한가지 속성이다. 이리하

여 두개의 명사가 남으며 결국 두개의 후보클라스 **Elevator**(승강기)와 **Button**(단추)가 남는다(UML습관은 클라스에는 굵은 서체(Boldface)를 리용하며 클라스이름의 첫 문자는 대문자로 쓰는것이다.).

최종클라스선도를 그림 12-4에 보여 주었다. 클라스 **Button**은 그림 12-2와 12-13의 대본들에서 사건 2, 4, 8, 11을 모형화하기 위하여 론리값속성 illuminated를 가진다. 이 문제는 두가지 류형의 단추를 명시하며 따라서 **Button**의 두개 부분클라스가 정의된다. 즉 **Elevator Button**과 **Floor Button**(열린 삼각형은 UML에서 계승을 의미한다.)이 정의된다. 매 **Elevator Button**과 **Floor Button**은 **Elevator**와 정보를 교환한다. 클라스 **Elevator**는 론리값속성 doors open을 가지는데 그것은 두개의 대본에서 사건 5, 9, 12, 14를 모형화한다.

유감스럽게도 출발이 잘되지 못하였다. 실지 승강기에서 단추들은 승강기와 직접 정보를 교환하지 않는다. 만일 어떤 특수한 요청에 의하여 어느 승강기를 보내야 하는가만을 결정하기 위하여서는 일정한 종류의 승강기조종기가 요구된다. 그러나 문제설정에서는 조종기에 대하여 전혀 언급하지 않고 있으므로 명사추출단계에서 그것은 클라스로 선택되지 않았다. 달리 말하면 클라스후보를 찾기 위한 이 절의 방법은 하나의 출발점을 주고 있지만 그이상의것은 기대하지 말아야 한다.

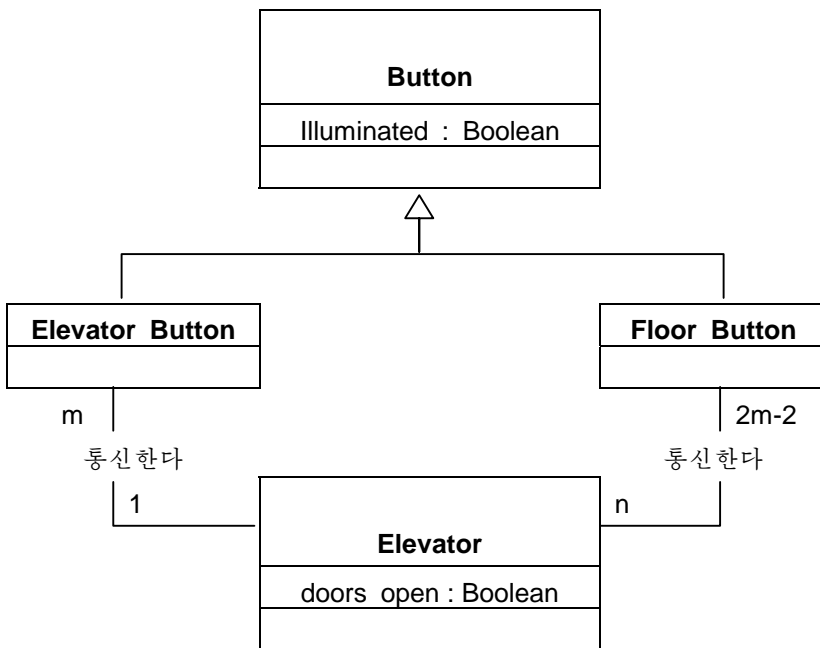



그림 12-4. 클라스도의 첫번째 반복

Elevator controller를 그림 12-4에 추가하여 그림 12-5를 얻는다. 이것은 확실히 합리적이다. 그러므로 임의의 시각에 지어는 통합단계만큼 늦어서도 클라스모형화에로 돌아 갈수 있다는것을 념두에 두면서 이 시점에서 3단계(동적모형화)에로 이행하는것이 합리적인것 같다. 그러나 동적모형화에로 나가기전에 클라스모형화를 위한 한가지 다른 기

법을 고찰한다.

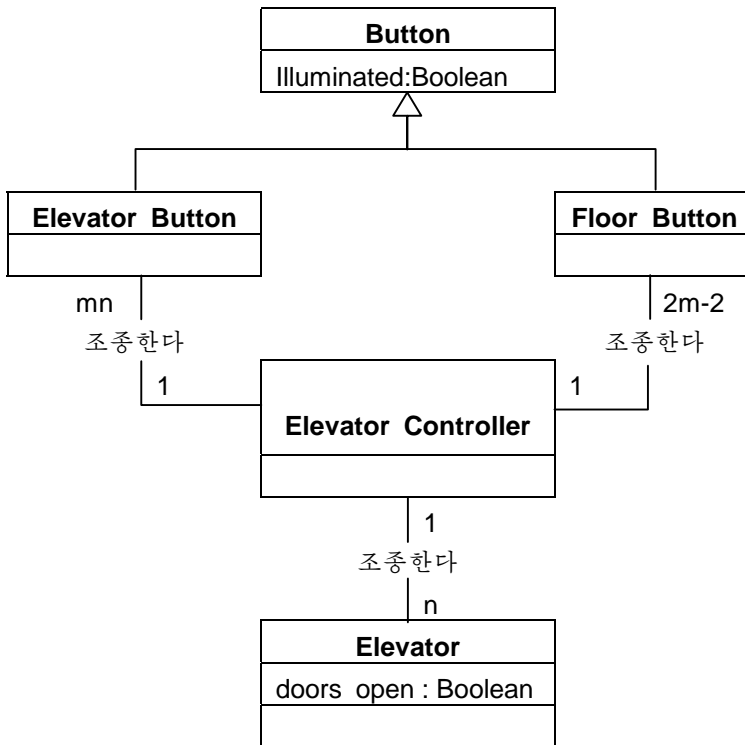



그림 12-5. 클라스도의 두번째 반복

12. 4. 2. CRC카드

오래동안 클라스-책임-협동(*class-responsibility-collaboration*(CRC))카드가 객체지향분석단계에서 리용되여 왔다[Wirfs-Brock, Wilkerson, and Wiener, 1990]. 매 클라스에 대하여 쏘프트웨어개발팀은 클라스의 이름, 그것의 기능(책임), 그 기능을 달성하기 위하여 호출하는 다른 클라스들의 목록(협동)을 보여 주는 하나의 카드를 작성한다. 이 방법은 그다음에 확장되였다. 첫째로 하나의 CRC카드는 흔히 자연언어로 표시된 클라스의 《책임》이 아니라 클라스의 속성과 방법들을 명백히 포함한다. 둘째로 기술이 변화되였다. 카드를 리용하는것 대신에 일부 개발기업체들은 부전지(*Post-it*)기록장에 클라스들의 이름을 붙인다. 개발기업체는 부전지기록장을 흰판우에서 돌리며 협동을 지적하기 위하여 부전지기록장들사이에 선이 그어 진다.

현재 이 전체 공정은 자동화될수 있다. 체계구성(*System Architect*)과 같은 CASE도구들은 화면상에서 CRC《카드》들을 창조하고 갱신하기 위한 모듈들을 포함하고 있다.

CRC카드의 우점은 다음과 같다. 개발팀이 CRC카드를 리용할 때 개발성원들사이의 호상작용은 어떤 클라스안에서의 오해나 부정확한 마당들 즉 속성이나 방법들을 뚜렷하

게 할수 있다. 또한 클라스들사이의 호상관계는 CRC카드가 리용될 때 뚜렷해 진다. 하나의 특별히 강력한 기법은 팀성원들속에 카드를 배포하는것이다. 그러면 그들은 자기의 클라스책임을 수행하게 된다. 이를 테면 누군가가 《나는 Date클라스인데 나의 책임은 새로운 날자객체를 창조하는것이다.》라고 말할수 있다. 그러면 다른 팀 성원은 자기는 Date클라스에서 추가적인 기능이 요구된다고 말할수 있다. 즉 어떤 날자를 습관적인 형식으로부터 하나의 옹근수 이를테면 1990년 1월 1일로부터의 일수로 변환하는것과 같은 기능, 결국 임의의 두 날자사이의 일수를 대응하는 두 옹근수를 더는것으로써 쉽게 계산할수 있도록 하는 기능을 요구할수 있다(다음의 《알고 싶은 문제》를 보시오.). 결국 CRC카드의 책임을 수행하는것은 클라스선도가 완전하고 정확하다는것을 검증하기 위한 효과적인 수단으로 된다.

알고 싶은 문제

1999년 2월 21일과 2001년 8월 10일사이의 일수를 어떻게 찾아 내겠는가? 이러한 덜기연산은 리자지불계산이나 미래의 현금류출입의 현재값결정과 같은 많은 재정계산들에서 요구된다. 이것을 위한 유용한 방법은 매 날자를 어떤 옹근수 즉 어떤 규정된 출발날자로부터의 일수로 변환하는것이다. 문제는 바로 리용할 출발날자를 일치시킬수 없다는것이다. 천문학자들은 율리우스날자 즉 기원전 4713년 1월 1일의 낮 12GMT(그리니치표준시간)로부터의 일수를 리용한다. 이 체계는 1582년에 조세프 스콜리저(Joseph Scaliger)에 의하여 발명되였는데 그는 이것을 자기 아버지 율리우스 스콜리저(Julius Caesar Scaliger)를 위하여 명명하였다(왜 기원전 4713년 1월 1일을 선택하였는가를 실지로 알고 싶다면 [USNO, 2000]에 문의하시오). 릴리안(Lilian)날자는 1582년 10월 15일 즉 로마법왕 그레고리 8세(Gregory XⅢ)가 도입한 그레고리력의 첫날로부터의 일수이다. 릴리안날자는 그레고리력개혁의 기본제안자인 루이지 릴리오(Luigi Lilio)를 위하여 명명하였다. 릴리오는 윤년법칙을 비롯한 그레고리력과 관련된 많은 알고리듬들을 개발하는데서 책임적역할을 하였다.

쏘프트웨어방면에서 고찰하면 ANCI COBOL85의 본질적기능들은 옹근수날자의 출발날자로서 1601년 1월 1일을 리용하고 있다. 그러나 거의 모든 표처리프로그람(*spreadsheets*)들은 Lotus 1-2-3의 안내를 따라 1900년 1월 1일을 리용하고 있다.

CRC카드의 약점은 이 방법이 일반적으로 팀성원들이 련관된 응용령역에 상당한 경험을 가지고 있지 않는 한 클라스들을 식별하기 위한 좋은 방법으로 되지 않는다는것이다. 한편 일단 개발자들이 많은 클라스들을 결정하고 그것들의 책임과 협동에 좋은 착상을 가지고 있다면 CRC카드는 그 공정을 완성하고 모든것이 정확하다는것을 확인하기 위한 훌륭한 방법으로 될수 있다. 이에 대하여서는 12.6에서 서술한다.

1 2. 5. 동적모형화

동적모형화의 목적은 매 클라스에 대하여 상태도(*state diagram*) 즉 유한상태기계에 류사한 목표제품에 대한 서술을 생성하는것이다. 우선 클라스 **Elevator Controller**를 고찰하자. 간단하게 하기 위하여 한대의 승강기만 고찰한다. **Elevator Controller**에 대한 상태도를 그림 12-6에 보여 주었다.

표시법은 11.6의 유한상태기계의 표시법과 약간 류사하지만 한가지 중요한 차이가 있다. 제11장에 제시된바와 같이 FSM은 형식적기법의 한가지 실레이다. 상태이행도 그자체는 작성될 제품에 대한 완전한 표현이 아니다. 그대신에 모형은 식 (11.2)로 주어지는 형식의 이행규칙들의 모임이다. 즉

current state and **event** and **predicate** ⇒ **next state**.

형식성은 수학규칙들의 모임형태로 모형을 제시함으로써 달성된다.

반대로 UML상태도의 표현은 약간 덜 형식적이다. 상태기계의 세개의 측면들(상태, 사건, 술어)은 UML선도우에 분포된다. 실레로 그림 12-6에서 상태 **Go into Wait State**는 만일 현재의 상태가 **Elevator Event Loop**이고 술어 [elevator stopped, no requests pending]이 옳다면 입력된다(UML술어는 그림 12-6에 보여 준것처럼 중괄호안에 나타난다.). 상태 **Go into Wait State**가 입력되였을 때 작용 Close elevator doors after timeout가 수행되게 된다. OOA현재판은 준형식적(도형)기법이며 따라서 상태도의 형식성의 본질적결여는 문제로 되지 않는다. 그러나 객체지향파라다임이 성숙될 때 보다 형식적인 판본이 개발되고 대응하는 동적모형들은 유한상태기계에 좀 더 가까와 질것 같다.

그림 12-6의 상태도와 그림 11-4부터 11-16까지의 STD의 등가성을 보기 위하여 각이한 대본들을 고찰하자. 실례로 그림 12-2의 대본의 첫 부분을 고찰하자. 먼저 사용자 A는 3층에서 상승층단추를 누른다. 만일 층단추가 조명이 켜지지 않았다면 그림 11-15와 그림 12-6의 상태 **Process Request**에 따라 그 단추는 필요할 때 켜진다. 상태도의 경우에 다음상태는 **Elevator Event Loop**이다.

그다음 승강기는 3층에 접근한다. 먼저 STD방법을 고찰하자. 그림 11-16에서 승강기는 상태 S(U, 3)에로 간다. 즉 승강기는 올라 가려는 3층에서 멈추어 선다(한대의 승강기뿐이라는 단순한 가정을 하였기때문에 그림 11-16에 있는 인수 e는 여기서 생략한다.). 그림 11-5로부터 승강기가 도착할 때 층 단추는 꺼진다. 이제(그림 11-6) 문이 닫기고 승강기는 4층을 향하여 움직이기 시작한다.

그림 12-6의 상태도에 돌아 와서 승강기가 3층에 접근할 때 일어 나는 상황을 고찰하자. 승강기는 움직임상태에 있기때문에 입력되는 다음상태는 **Determine if Stop Requested**이다. 요청은 접수되며 사용자 A는 승강기를 여기서 멈출것을 요청하였기때문에 다음상태는 **Stop at Floor**이다. 승강기는 3층에서 멈추어 서고 문이 열린다. 3층 승강기단추가 눌리워 지지 않았으므로 다음 상태는 **Elevator Event Loop**이다.

사용자 A는 승강기에 오르고 7층 단추를 누른다. 다음상태는 다시 **Process Request**이고 다음은 다시 **Elevator Event Loop**이다. 승강기는 멈추어섰고 두가지 요청이 해결되지 않은채로 있으므로 다음상태는 **Close Elevator Doors**이며 문은 설정시간이 지나면 닫긴다. 사용자 A가 3층단추를 눌렀기때문에 **Floor Button Off**가 다음상태로 되며 층단추

는 꺼진다. 그 다음상태는 **Process Next Request**이며 승강기는 4층을 향하여 움직이기 시작한다. 대응하는 선도들의 련관된 측면들은 명백히 이 대본에 관하여 등가이다. 물론 다른 대본들을 고찰하기를 원할수도 있다.

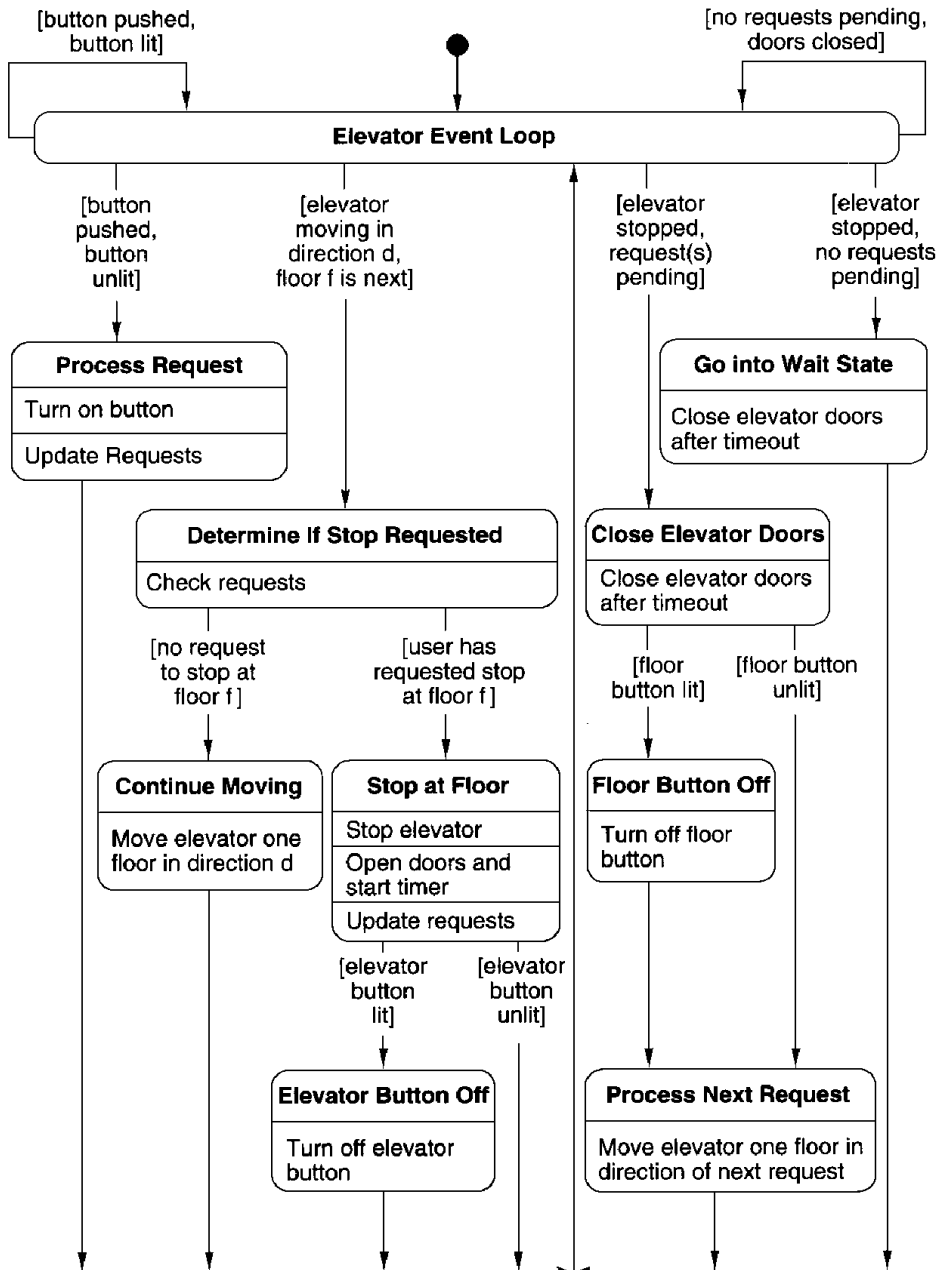



그림 12-6. 승강기조종기클라스를 위한 상태도의 첫번째 반복

앞선 론의로부터 그림 12-6이 그 대본들로부터 구성되였다고 하는데 놀랄 필요가

없다. 보다 정확하게는 대본들의 특정한 사건들은 일반화되였다. 실례로 그림 12-2의 대본의 첫 사건 즉 《사용자 A는 3층에서 상승층단추를 누른다.》를 고찰하자. 이 특정한 사건은 어떤 임의의 단추를 누르는것으로 일반화된다. 그러면 두가지 가능성이 존재한다. 즉 그 단추가 이미 켜졌든가(이 경우에는 아무 상황도 일어 나지 않는다.) 그 단추가 켜지지 않았든가(이 경우에는 사용자의 요청을 처리하기 위한 작용이 취해 져야 한다.)이다.

이 사건을 모형화하기 위하여 **Elevator Event Loop**상태가 그림 12-6에 그려 진다. 이미 조명이 켜진 단추의 경우는 그림 12-6의 웃단 왼쪽 구석에 있는 경계상태 [button pushed, button unlit]로서 아무것도 하지 않는 고리로 모형화된다. 다른 경우에 조명이 꺼진 단추는 경계상태 [button pushed, button unlit]로서 표식 붙은 상태 **Process Request**로 이끌어 가는 화살표로 모형화된다. 대본의 사건 2로부터 작용 Turn on button이 이 상태에서 요구된다는것은 명백하다. 더우기 임의의 단추를 누르는것과 관련한 사용자의 작용 목적은 한대의 승강기를 요청하는것이며 따라서 작용 Update requests가 또한 상태 **Process Request**에서 수행되여야 한다.

이때 대본의 사건 3. **승강기가 3층에 도착한다**를 고찰하자. 이것은 층사이에서 움직이는 임의의 승강기라는 개념으로 일반화되였다. 승강기의 이동은 경계상태[elevator moving in direction d, floor f is next]와 상태 **Determine if Stop Requested**로 모형화된다. 그러나 두개의 가능성이 존재한다. 즉 승강기를 f층에서 멈추려는 요청이든지 그러한 요청이 없든지이다. 앞의 경우에 경계상태[no request to stop at floor f]에 대응하여 승강기는 d방향으로 한층 더 계속 움직여야(**Continue Moving**) 한다. 후자의 경우에(경계상태[user has requested stop at floor f]에 대응하여) 그림 12-2의 대본으로부터 승강기를 멈추(Stop elevator)고(사건 3으로부터) 그다음 문을 열고 시간계수기를 동작(Open doors and start timer)시키는것(사건 5와 6으로부터)이 필요하다는것은 명백하다. 또한 **Precess Request**상태의 경우와 류사하게 상태 **Stop at Floor**에서 요구를 갱신(Update requests)하는것이 필요하다는것이 명백하다. 마지막으로 대본의 사건 4를 일반화하는것은 만일 승강기단추가 켜졌으면 그것을 꺼야 한다는것을 실현하는데로 이끌어 간다. 이것은 상태 **Elevator Button Off**(그림 12-6의 하단 왼쪽에 있는 상태)와 이 상태를 표현하고 있는 경우의 2개의 경계상태들에 의하여 모형화된다.

그림 12-2의 대본의 사건 9를 모형화하면 상태 **Close Elevator Doors**가 생성된다. 사건 10을 모형화하면 상태 **Process Next Request**가 생성된다. 그러나 상태 **Go into Wait State**와 경계상태[no requests pending, doors closed]에 대한 요구는 다른 대본의 사건 즉 사용자가 승강기에서 내리고 조명이 켜져 있는 단추가 하나도 없는 사건을 일반화하여 유도된다.

다른 클라스들에 대한 상태도는 상대적으로 단순하므로 련습문제로 남겨 둔다(문제 12.1).

12. 6. 객체지향분석단계에서의 시험

이 시점에서 객체지향분석공정의 세개의 모형이 완성될수 있다. 다음단계는 지금까지의 OOA를 검토하는것이다. 이 검토의 한가지 요소는 12.4.2에서 제안한것처럼 CRC카

드를 리용하는것이다.

따라서 매 클라스 **Button, Elevator Button, Floor Button, Elavator, Elevator Controller**에 대하여 CRC카드가 기입된다. 그림 12-7에 보여 준 **Elevator Controller**에 대한 CRC카드는 그림 12-5의 클라스도와 그림 12-6의 상태도로부터 연역된다.

보다 자세히 고찰하면 **Elevator Controller**의 책임(RESPONSIBILITY)은 **Elevator Controller**에 대한 상태도에 있는 모든 작용들을 렬거함으로써 얻어 졌다(그림 12-6). **Elevator Controller**의 협동(COLLABORATION)은 그림 12-5의 클라스도를 시험하고 클라스 **Elevator Button, Floor, Button, Elevator**가 클라스 **Elevator Controller**와 호상작용한다는것을 기록함으로써 결정되였다.

이 CRC는 객체지향분석의 첫 반복과 함께 두가지 중요한 문제들을 강조하고 있다. 먼저 책임 1. **승강기단추를 켠다**를 고찰하자. 이 지령은 객체지향파라다임에서 총적으로 어울리지 않는다. 책임구동설계의 관점(1.6)으로부터 클라스 **Elevator Button**의 객체들은 그 단추들을 켜거나 끄는데 책임이 있다. 또한 정보은폐의 관점으로부터 **Elevator Controller**는 어떤 단추에 조명을 켜거나 끄는데 필요한 **Elevator Button**의 내부에 관한 지식을 가지지 말아야 한다.

<table>
<tr><th colspan="2">클 라 스
Elevator Controller</th></tr>
<tr><td colspan="2">책 임</td></tr>
<tr><td>1</td><td>승강기단추를 켠다.</td></tr>
<tr><td>2</td><td>승강기단추를 끈다.</td></tr>
<tr><td>3</td><td>층단추를 켠다.</td></tr>
<tr><td>4</td><td>층단추를 끈다.</td></tr>
<tr><td>5</td><td>승강기를 한 층 우로 이동시킨다.</td></tr>
<tr><td>6</td><td>승강기를 한 층 아래로 이동시킨다.</td></tr>
<tr><td>7</td><td>승강기문을 열고 시간계수기를 동작시킨다.</td></tr>
<tr><td>8</td><td>설정시간이 지나면 승강기문을 닫는다.</td></tr>
<tr><td>9</td><td>요구를 검사한다.</td></tr>
<tr><td>10</td><td>요구를 갱신한다.</td></tr>
<tr><td colspan="2">협 력</td></tr>
<tr><td>1</td><td>클라스 Elevator Button</td></tr>
<tr><td>2</td><td>클라스 Floor Button</td></tr>
<tr><td>3</td><td>클라스 Elevator</td></tr>
</table>

그림 12-7. 클라스 Elevator Controller를 위한 CRC카드의 첫번째 반복

정확한 책임은 다음과 같다. 즉 승강기단추에 불을 켜기 위하여 **Elevator Button**에 통보를 보내는것이다. 그림 12-7에 있는 책임 2부터 6까지에 관하여 이와 류사한 변경들이

필요하다. 이 6개의 변경들은 그림 12-8 **Elevator Controller**에 대한 CRC카드의 두번째 반복에서 반영된다.

두번째 문제는 클라스가 검열되였다는것이다. 책임 7. **승강기문을 열고 시간계수기를 동작시킨다**를 고찰하자. 여기서 중요한 개념은 상태(*state*)에 대한 개념이다. 어떤 클라스의 속성들을 때때로 상태변수(*state variable*)라고 부른다. 그 리유는 대부분 객체지향실천들에서 제품의 상태가 각이한 요소객체들의 속성값에 의하여 결정되기때문이다. 상태도는 유한상태기계와 공통적인 많은 특성을 가지고 있다. 따라서 상태의 개념이 객체지향파라다임에서 중요한 역할을 논다는것은 놀라운 일이 아니다. 이 개념은 어떤 요소가 클라스로 모형화되여야 하는가를 결정하는것을 돕는데 리용될수 있다. 만약 문제로 되는 요소가 실현의 실행기간에 변화된 상태를 가진다면 그것은 아마도 클라스로 모형화되여야 한다. 명백히 승강기의 문은 하나의 상태(열거나 닫는)를 가지며 그러므로 **Elevator Doors**는 하나의 클라스로 되여야 한다.

Elevator Doors가 클라스로 되여야 할 또 하나의 리유가 있다. 객체지향파라다임은 상태가 객체내에서 은폐되여 허가되지 않은 변경으로부터 보호되도록 한다. 만일 하나의 **Elevator Doors**객체가 있다면 승강기문을 열거나 닫을수 있는 유일한 방도는 이 **Elevator Doors**객체에 통보를 보내는것이다. 제정되지 않은 시간에 승강기문을 열거나 닫으면 엄중한 사고가 발생할수 있다(즉 다음의 《알고 싶은 문제》를 보시오.). 그러므로 일정한 류형의 제품들에 대하여 7장과 8장에서 렬거된 객체의 기타 우월성에 안정성제약을 추가하여야 한다.

알고 싶은 문제

몇년전에 내가 한 건물의 10층에서 조바심이 나서 승강기를 기다리고 있었다 문이 열리고 나는 앞으로 걸음을 떼기 시작하였다. 그러나 승강기는 거기에 없었다. 내가 승강기통로에로 걸어 가려다가 멈춰 서게 된것은 앞에서 완전한 어둠을 보았기때문이다 본능적으로 무엇인가 잘못 되였다는것을 깨달았기때문이다.

아마 승강기조종체계가 객체지향파라다임을 리용하여 개발되였다면 10층에서의 부적당한 문열기와 같은 일은 없었을수도 있었다.

Elevator Doors를 클라스에 넣는다는것은 그림 12-7에 있는 책임 7과 8이 책임 1부터 6까지와 류사하게 변경될 필요가 있다는것을 의미한다.

즉 문을 열거나 닫기 위하여 **Elevator Doors**에 통보들을 보내야 한다. 그러나 추가적인 복잡성이 생긴다. 책임 7은 **승강기문을 열고 시간계수기를 동작시키라**이다. 이것은 두개의 개별적인 책임으로 갈라야 한다. 한가지 통보가 승강기문을 열기 위하여 **Elevator Doors**에 보내져야 한다. 그러나 시간계수기는 승강기조종기 **Elevator Controller**의 한 부분이며 그렇기때문에 시간계수기를 동작시키는것은 **Elevator Controller** 그자체의 책임이다. **Elevator Controller**에 대한 CRC카드의 두번째 반복(그림 12-8)은 이와 같은 책임의 분리가 성과적으로 수행되였다는것을 보여 주고 있다. 그림 12-7의 CRC카드에 의해 강조된 두개의 중요한 문제외에도 **Elevator Controller**의 책임 check requests와 update requests들

은 속성 requests를 클라스 **Elevator Controller**에 추가할것을 요구한다. 이 단계에서 requests는 간단히 requestType형이 되도록 정의된다. 즉 requests에 대한 자료구조는 상세설계단계에서 선택될것이다.

클 라 스 **Elevator Controller**	
책 임	
1	승강기조종단추에 조종단추를 켜도록 통보문을 보낸다.
2	승강기조종단추에 조종단추를 끄도록 통보문을 보낸다.
3	층조종단추에 조종단추를 켜도록 통보문을 보낸다.
4	층조종단추에 조종단추를 끄도록 통보문을 보낸다.
5	승강기에 한 층 우로 이동하도록 통보문을 보낸다.
6	승강기에 한 층 아래로 이동하도록 통보문을 보낸다.
7	승강기문을 열도록 통보문을 보낸다.
8	시간계수기를 동작시킨다.
9	설정시간이 지나면 승강기문을 닫으라고 통보문을 보낸다.
10	요구를 시험한다.
11	요구를 갱신한다.
협 력	
1	클라스 **Elevator Button**
2	클라스 **Floor Button**
3	클라스 **Elevator**

그림 12-8. 클라스 **Elevator Controller**를 위한 CRC카드의 두번째 반복

수정된 클라스를 그림 12-9에 보여 주었다. 클라스도를 변경하면서 유스케스와 상태도들은 이후의 세련을 필요로 하는가를 보여 줄수 있도록 재시험되여야 한다.

유스케스도는 명백히 이 경우에도 적절하다. 그러나 그림 12-6의 상태도의 작용들은 그림 12-8(CRC카드의 두번째 반복)의 책임은 반영하고 그림 12-7(첫번째 반복)의 책임은 반영하지 않도록 변경되여야 한다. 또한 상태도들의 모임은 추가적인 클라스들을 포함하도록 확장되여야 한다. 대본들은 이 변경들을 반영하도록 갱신될 필요가 있다. 즉 그림 12-10은 그림 12-2의 대본에 대한 두번째 반복을 보여 준다. 지어 이 모든 변경들이 만들어 지고 검사된후에도(변경된 CRC카드를 비롯하여) 객체지향분석에로 돌아 와 하나이상의 선도들을 재검사하는것이 객체지향설계단계에서 여전히 필요할수도 있다.

11.1에서 지적한바와 같이 명세서는 의뢰자와 개발자사이의 계약서이다. 따라서 OOA가 완성된후에 보다 습관적인 명세서가 작성되여야 한다. 즉 한가지 대책안은 부록 4의 것과 류사한 형식을 리용하는것이다. 일단 허가되면 명세서를 각이한 UML선도와 함께 설계팀에 넘겨 준다.

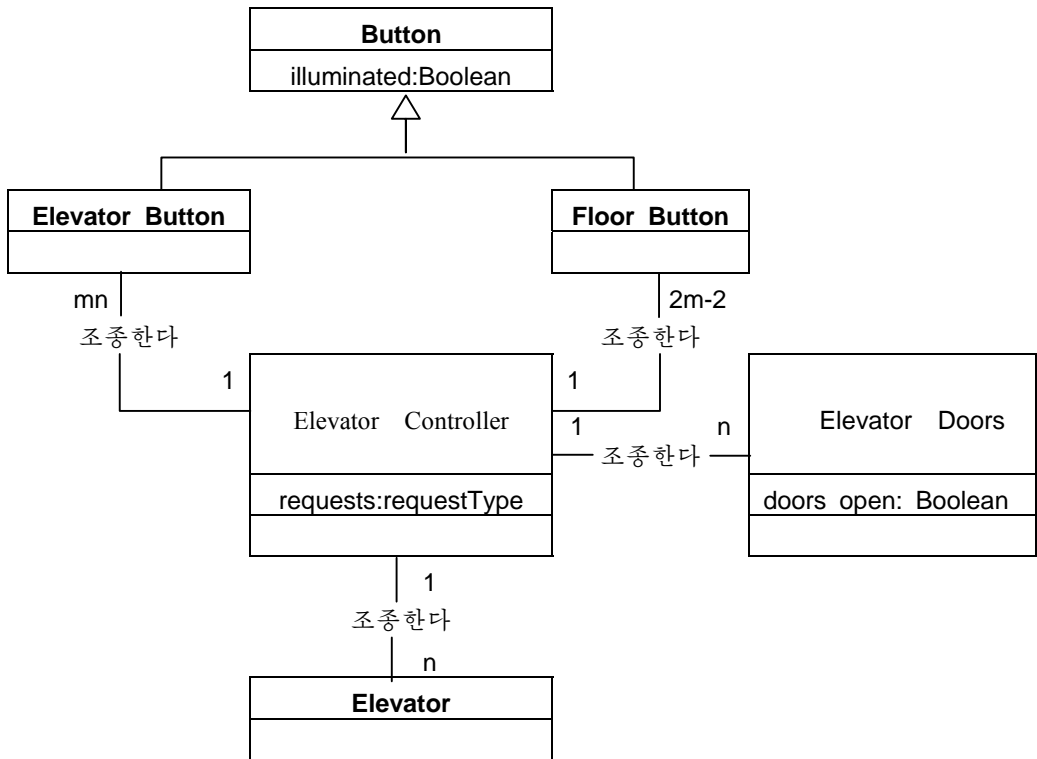



그림 12-9. 클라스도의 세번째 반복

1. 사용자 A는 3층에서 상승층단추를 눌러 승강기를 요청한다.
   사용자 A는 7층으로 가려고 한다.
2. 층단추는 승강기조종기에 층 단추가 눌러 졌다는것을 통보한다.
3. 승강기조종기는 상승층 단추가 켜지도록 상승층 단추에 통보를 보낸다.
4. 승강기조종기는 승강기가 3층으로 이동하도록 승강기에 일련의 통보들을 보낸다. 승강기에는 사용자 B가 있는데 그는 1층에서 승강기에 올라 9층 승강기단추를 눌렀다.
5. 승강기조종기는 상승층단추가 꺼지도록 상승층단추에 통보를 보낸다.
6. 승강기조종기는 승강기문을 열도록 승강기문에 통보를 보낸다.
7. 승강기조종기는 시간계수기를 동작시킨다.
   사용자 A가 승강기에 오른다.
8. 사용자 A는 7층 승강기단추를 누른다.
9. 승강기조종기는 승강기단추가 눌러 졌다는것을 통보한다.
10. 승강기조종기는 7층승강기단추가 켜지도록 그 단추에 통보를 보낸다.
11. 승강기조종기는 설정시간이 지나면 승강기문을 닫도록 승강기문에 통보를 보낸다.

그림 12-10. 표준대본의 두번째 반복

12. 승강기조종기는 승강기가 7층으로 이동하도록 승강기에 일련의 통보들을 보낸다.
13. 승강기조종기는 7층승강기단추가 꺼지도록 그 단추에 통보를 보낸다.
14. 승강기조종기는 사용자 A가 승강기에서 내리도록 승강기문을 열기 위하여 승강기문에 통보를 보낸다.
15. 승강기조종기는 시간계수기를 동작시킨다.
    사용자 A는 승강기에서 내린다.
16. 승강기조종기는 설정시간이 지나면 승강기문을 닫도록 승강기문에 통보를 보낸다.
17. 승강기조종기는 사용자 B를 태우고 승강기가 9층으로 이동하도록 승강기에 일련의 통보들을 보낸다.

그림 12-10. 표준대본의 두번째 반복(계속)

12. 7. 객체지향분석단계를 위한 CASE도구

객체지향분석에서 선도가 노는 역할을 생각하면 많은 CASE도구들이 개발되여 객체지향분석을 지원하고 있다는것은 놀라운 일이 아니다. 그 기본형식에서 이와 같은 도구는 본질상 매 모형화단계를 쉽게 수행할수 있게 하는 그리기도구이다. 보다 중요하게는 손으로 그린 도형을 변경시키려고 하는것보다 그리기도구로 작성된 선도를 변경시키는것이 훨씬 더 쉽다. 결국 이러한 류형의 CASE도구는 객체지향분석의 도형측면을 지원하고 있다.

더우기 이러한 류형의 일부 도구들은 모든 련관된 선도들뿐아니라 CRC카드도 그린다. 이 도구들의 우점은 기초를 이루는 모형에 대한 변경이 영향을 받은 모든 선도들에 자동적으로 반영된다는것이다. 결국 각이한 선도들은 순전히 기초를 이루는 모형에 대한 서로 다른 관점들로 된다.

한편 일부 CASE도구들은 객체지향분석뿐아니라 객체지향생명주기의 나머지 상당한 부분들을 지원한다. 현재 이 모든 도구들은 사실상 UML을 지원하고 있다[Rumbaugh, Jacobson, and Booch, 1999]. 이와 같은 도구들의 실례들은 Rose and Together를 포함하고 있다.

12. 8. 항공음식전문회사 실례연구 : 객체지향분석

객체지향분석은 세개의 단계 유스-케스모형화, 클라스모형화, 동적모형화로 이루어진다. 이 단계들은 첫단계 즉 유스-케스모형화로부터 시작하여 반복적으로 수행된다. 항공음식전문회사 신속원형의 주차림표(부록 3 또는 4)는 유스케스의 목록을 생성한다. 이것들은 또한 상당히 많은 품을 들여서 10.14의 요구들로부터 직접 연역될수 있었다.

어떤 예약을 하기 위한 유스케스를 그림 12-11에 보여 주었다. 이 유스케스선도에 대하여 표준 및 례외대본들이 요구된다. 한가지 방법은 개별적인 표준 및 례외대본들을 구성하는

것이다. 승강기문제에 관하여 이 과정이 수행되였다(그림 12-2와 12-3). 항공음식전문회사문제에서 우리는 약간 다른 형식화 즉 가능한 대용품들을 반영하는 확장된 대본을 리용한다. 그림 12-11에 대응하는 확장된 대본은 그림 12-12에 보여 주었다.

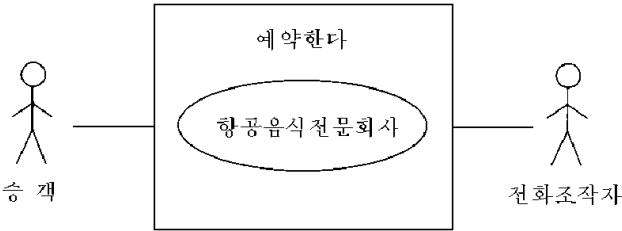



그림 12-11. 항공음식전문회사제품의 예약을 위한 유스-케스도

1. 승객은 항공음식전문회사호출쎈터 교환수에게 전화를 걸어 비행기를 예약하려는 희망을 표현한다. 2. 교환수는 승객의 이름과 주소를 요구한다. 3. 승객은 요구하는 정보를 제공한다. 4. 교환수는 승객에게 비행날자와 비행기번호를 묻는다. 5. 승객은 요구하는 정보를 제공한다. 6. 교환수는 승객에게 특별식사가 요구되는가를 묻는다. 7. 승객은 어떤 종류의 특별식사가 요구되는가 하는것을 설명한다. 8. 교환수는 승객에게 좌석선택에 대하여 묻는다. 9. 승객은 자기의 좌석선택을 제공한다. 10. 교환수는 지불정보를 요구한다. 11. 승객은 유효신용카드번호를 제공한다. 12. 교환수는 이 예약을 항공음식전문회사자료기지에 위탁한다. 13. 항공음식전문회사자료기지는 예약식별번호(reservation ID)를 교환수에게 발행한다. 14. 교환수는 예약식별번호를 승객에게 주고 승객에게 표가 곧 우편으로 보내질것이라는것을 통보한다.
가능한 대안
ㄱ) 요청한 비행기번호가 승객이 비행하려는 특정한 날에 존재하지 않는다. ㄴ) 승객이 요청한 비행기는 이미 만원이다. ㄷ) 승객은 항공음식전문회사가 충족시킬수 없는 류다른 식사를 요청한다. ㄹ) 승객이 제공한 신용카드에 문제가 있다.

그림 12-12. 그림 12-11에 대응하는 확장된 대본

그림 12-12는 원형(부록 3 또는 4)으로부터 구성되였다. 주차림표에서 Enter a

Reservation(그림 12-12에서 사건 1로 표시된다)을 선택하면 비행정보(사건 4와 5), 좌석선택(사건 8과 9), 특별식사정보(사건 6과 7), 승객정보(사건 2와 3)에 대한 요청이 발생한다. 신속원형은 좋지 못한 순서로 정보를 요청하며 예약에 대한 지불이 생략되였다는것을 곧 깨닫게 된다. 요청들을 재배렬하고 지불부분을 추가하여 그림 12-12의 사건 1로부터 11까지를 생성하였다. 대본의 마지막 3개의 사건들은 대본의 표준부분을 완성하기 위하여 추가되였다.

가능한 대안은 표준사건들에 대하여 작업하는 동안 연역되였다. 신속원형에서 가능한 대안을 찾는것은 좋은 생각이 아니다. 신속원형은 포함하고 있는 코드를 고찰하기 위하여 너무 조급하게 구성되여서 그림 12-12에 있는 사건 ㄱ)부터 ㄹ)까지와 같은 대안들을 다룰수 없다.

우편엽서를 되돌려 보내고 조사하기 위한 유스케스를 그림 12-13에 보여 주었다. 그리고 대응하는 확장대본은 그림 12-14에 보여 주었다. 이 대본의 사건 3은 신속원형의 엽서조사부분으로부터 얻어 졌다. 사건 1과 2는 그다음 그 대본을 완성하기 위하여 추가되였다. 앞의 대본에서와 마찬가지로 4개의 가능한 대안들이 표준사건들에 대하여 작업하는 동안 연역되였다.

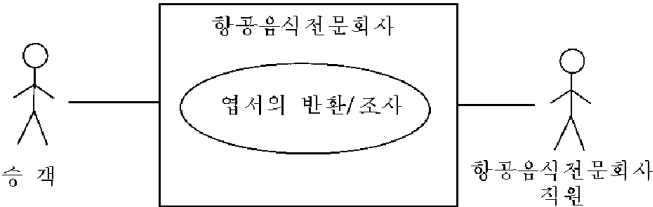



그림 12-13. 엽서를 반환 및 조사하는 유스-케스도

<table>
<tr><td>
1. 비행기가 착륙한 다음 항공음식전문회사 직원은 특별식사를 접수한 승객에게 평가카드를 우편으로 보낸다.
2. 승객은 엽서를 받고 식사를 평가한 다음 엽서를 항공음식전문회사에 보낸다.
3. 항공음식전문회사 직원은 엽서를 접수하고 항공음식전문회사 자료기지 안의 적당한 비행기록을 갱신하며 승객의 대답을 기록 한다.
</td></tr>
<tr><td>가능한 대안</td></tr>
<tr><td>
ㄱ) 승객은 엽서를 접수하지 않는다.
ㄴ) 승객은 엽서를 돌려 보내지 않는다.
ㄷ) 승객은 엽서를 틀리게 표식한다. 실례로 타당한 값범위밖에서 답변을 준다.
ㄹ) 특별식사를 요청하지 않은 승객이 엽서를 받고 그것을 우편으로 되돌려 보낸다
</td></tr>
</table>

그림 12-14. 그림 12-13에 대응하는 확장된 대본

기타 유스케스들(특별식사목록의 참조와 선택, 특별식사의 수송, 《저염식사》보고서보기, 《퍼센트통계》보고서보기, 《오르지 않은 식사》보고서보기, 《저질식사》보고서보기)은 그 개별적인 확장대본들과 류사하다. 그러므로 간결하게 하기 위하여 그것들을 여기서 생략한다.

두번째 단계는 클라스모형화(*class modeling*)이다. 이 단계의 목적은 클라스들을 추출하고 클라스의 속성들을 찾고 그것들의 호상관계와 호상작용들을 결정하는것이다. 이것은 대본 또는 CRC카드가 아니라 12.4의 세단계 명사추출방법을 리용하여 수행되였다. 1단계는 제품을 될수록 간단하게 명확하게 단일문장으로 정의하는것이다. 항공음식전문회사의 경우에 이를 위한 한가지 가능한 방법은 다음과 같다.

> 콤퓨터화된 체계가 특별식사프로그람의 효과성을 고려하여 정보를 제공하기 위하여 요구된다.

2단계에서 비형식적전략이 단일단락으로 작성된다. 한가지 가능한 단락은 다음과 같다.

> 보고서가 특별식사프로그람의 효과성을 문서화하기 위하여 생성되게 된다. 이 보고서들은 비행기에 오른 식사, 승객이 오른 비행기, 승객들의 이름과 주소, 식사질, 저염식사질과 관련되여 있다.

3단계에서 이 단락안에 있는 명사들이 식별되여 다음과 같은 단락이 생성된다.

> **보고서**가 특별식사**프로그람**의 **효과성**을 문서화하기 위하여 생성되게 된다. 이 보고서들은 비행기에 오른 식사와 승객들의 비행기탑승의 **퍼센트**, **승객**들의 **이름**과 **주소**, 식사**질**, 저염식사**질**과 관련되여 있다.

명사들은 보고서(report), 효과성(efficacy), 프로그람(program), 퍼센트(percentage), 승객(passenger), 이름(name), 주소(address), 질(quality)이다. 효과성(efficacy), 프로그람(program), 퍼센트(percentage), 탑승(boarding), 질(quality)은 추상명사들이며 그렇기때문에 클라스로 될것같지 않다. 이름(Name)과 주소(address)는 명확히 승객(passenger)의 속성들이다. 주어진 비행기에서 특별식사의 존재와 결여는 중심적인 론점이다. 명백치 않는것은 식사(meal) 그자체 또는 비행(flight) 그자체가 클라스로 되겠는가 하는것이다. 클라스를 추가하는것은 삭제하는것보다 일반적으로 더 쉽기때문에 적어도 첫번째 반복에서는 이 두 명사를 제쳐 놓는것이 더 좋았을것이라고 느껴 진다. 이로부터 두개의 명사들이 남게 되며 따라서 두개의 후보클라스 즉 **Report**와 **Passenger**가 생성된다. 최종적인 클라스도의 첫 반복은 그림 12-15에 보여 주었다. 이 그림은 네개의 부분클라스를 포함하여 두개의 후보클라스를 반영하고 있는데 매 보고서에 하나의 부분클라스가 할당된다. **Report**클라스는 네개의 보고서에 공통인 마당 즉 시작날자와 마감날자를 포함한다. 매 부분클라스는 그 보고서에 특정한 마당들을 포함한다.

이 클라스도에 여러가지 문제들이 존재한다.

1. 각이한 보고서들을 인쇄하기 위하여 요구되는 정보를 계산하자면 매번 비행할 때마다 자료가 요구된다. 실례로 특별식사를 주문한 비행기에 오른 승객들의 퍼센트를 결정하기 위하여서는 자료를 비행기안에 조직해 넣어야 한다.
2. 매 보고서는 여러개의 비행기록들을 호출하여야 하며 매 비행은 여러명의 승객들

을 가진다.

3. 10.14은 그림 12-15에 반영된것처럼 네개의 번호 붙은 보고서를 명시하고 있다. 그러나 첫번째 번호 붙은 보고서는 세개의 개별적인 보고서들로 이루어 진다. 그러므로 클라스도는 **Report**클라스의 6개(4개가 아니라)의 부분클라스들을 반영하여야 한다.

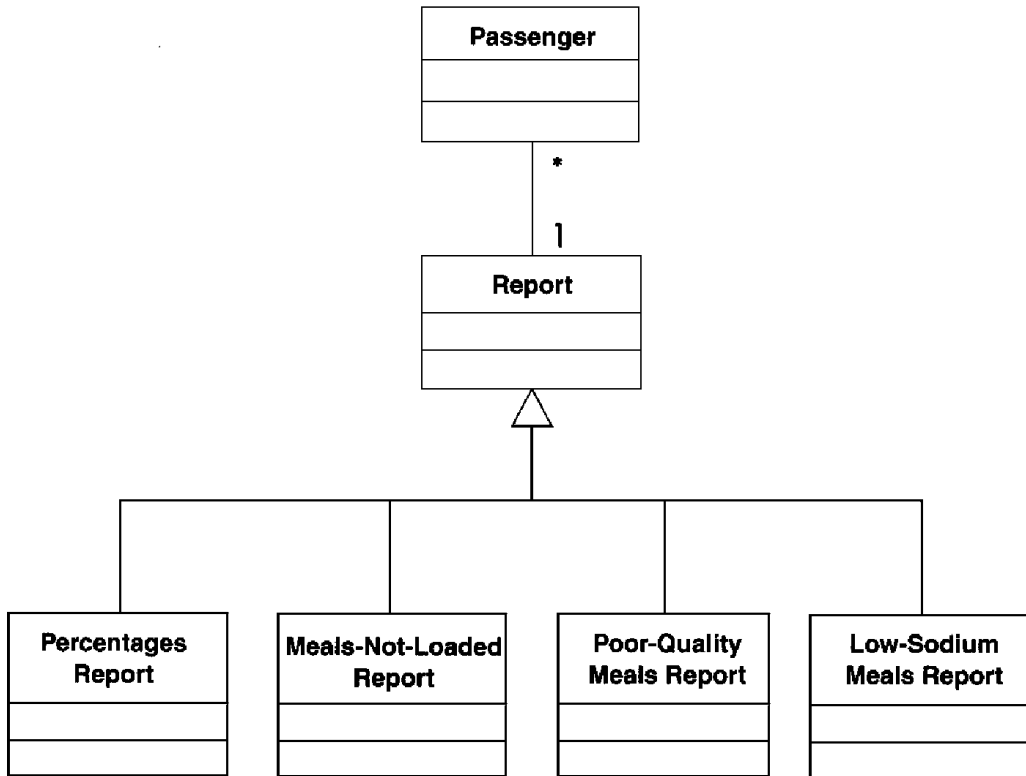



그림 12-15. 항공음식전문회사제품을 위한 클라스도의 첫번째 반복

이 사건들이 그림 12-16의 클라스도에 반영된다. 거기에는 실지로 **Report**의 6개의 부분클라스가 있다. 더우기 **Report**와 **Flight Record**사이에는 1대 n 관계가 존재한다. 이것은 매 보고서가 여러 비행기들로부터 얻은 정보들을 포함하고 있다는것을 반영하고 있다. 결국 **Flight Record**열에 열려 있는 다이야몬드표식은 집합을 의미하여(7.7) **Passenger**열의 *기호는 1대 n 관계를 의미한다. 즉 매 비행기록은 여러개의 승객기록들을 포함한다. 이 기록들은 **Flight Record**의 요소들로써 모형화되였다. 그대신 단순성과 명백성을 위하여 **Passenger**클라스는 독립적인 실체로서 취급된다. 마당(열쇠) passenger ID는 주어 진 **Passenger**를 대응하는 **Flight Record**와 련관시키기 위하여 리용된다.

클라스도의 첫번째 반복은 무엇때문에 그렇게 틀렸는가? 이 문제의 한가지 원인은 Flight를 하나의 후보속성으로 포함하지 않도록 결정한데 있다. 누구나 정상적인 기준을 가지고 있는데 이때에는 Flight도 Meal도 초기후보클라스로 선택하지 않은데 원인이 있는것 같다.

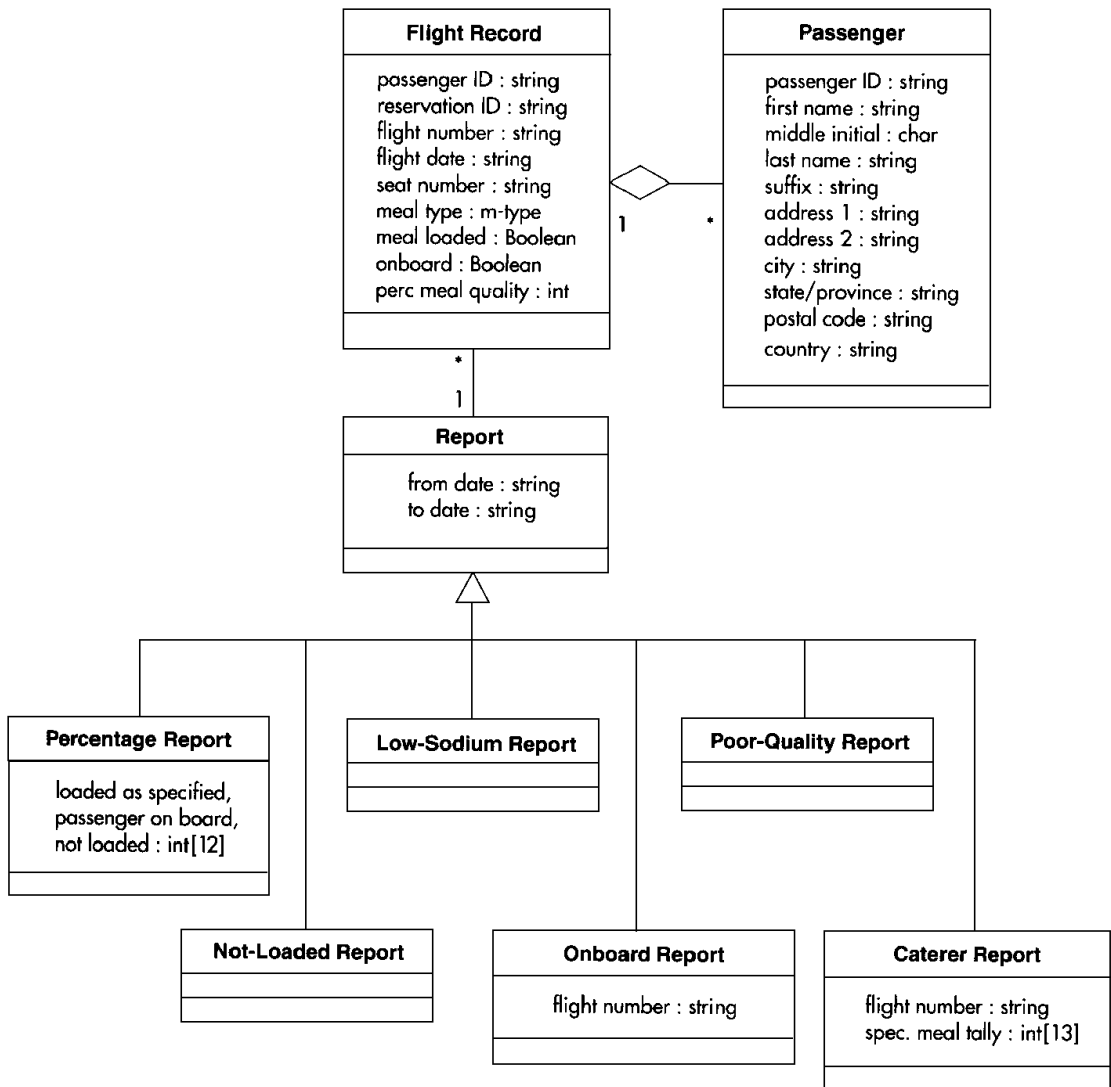



그림 12-16. 항공음식전문회사제품을 위한 클라스도의 두번째 반복

앞에서 지적한바와 같이 클라스를 추가하기는 쉽지만 그것을 제거하기는 훨씬 더 어렵다. 그러므로 첫번째 반복에서 두개의 후보클라스만 선택함으로써 얻은 리득이 잃는것보다 더 많았다.

네개의 보고서부분클라스들만 병합하기로 한 결정을 정당화하는것은 그리 쉽지 않다. 결국 신속원형(부록 3 또는 부록 2)은 6개의 보고서를 포함한다. 한편 이것은 부차적인 론의이며 쉽게 해결된다.

첫번째 반복의 부적당성을 념려하는것(그림 12-15)대신에 일단 보고서에 대한 자료가 비행기의 승객이 아니라 비행에 기초하여 조직되여야 한다는것을 깨달았으면 두번째 반복

이 쉽게 생성된다는것을 지적하는것이 더 좋았을것이다. 달리 말하면 반복이 객체지향파라다임에 대하여서는 일반적으로 본질적이라면 객체지향분석에 대하여서는 특별히 본질적이다. 반복은 오유가 발견되였을 때 어떤 제품을 버리는것이 아니라 오히려 그 제품을 계단식으로 변경하여 오유를 정정해 나간다는것을 의미한다.

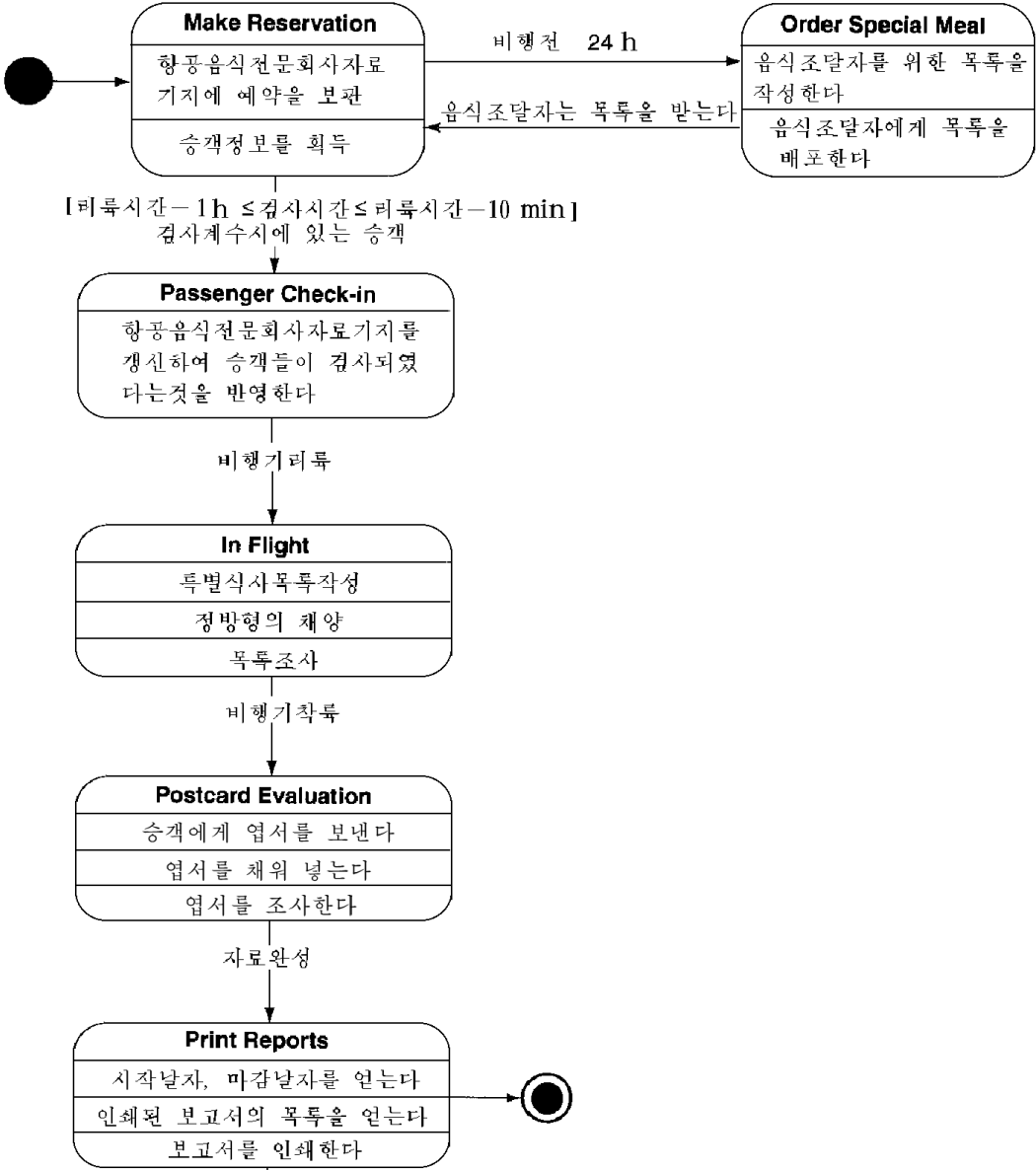



그림 12-17. 항공음식전문회사제품를 위한 상태도

객체지향분석에서 세번째 단계는 동적모형화이다. 완전한 제품에 대한 모든 대본들은 그림 12-17의 상태도에 반영되여 있다. 엄격히 말하여 하나의 상태도는 전체적으로

제품이 아니라 하나의 클라스에 관계하고 있다. 그러나 항공음식전문회사제품의 클라스들이 상태로부터 상태에로 옮겨 가지 않기때문에 그림 12-17과 같은 상태도는 항공음식전문회사제품에 보다 적합하다. 이것으로 객체지향분석이 완성된듯 하다. 그러므로 그것을 검토하여야 한다. 의뢰자도 개발팀도 UML선도들의 모임을 적당한 명세서로 기꺼이 받아 들이려 하지 않는다. 결국 11.1에서 지적한바와 같이 명세서는 의뢰자와 개발자들사이의 계약서이다. 따라서 OOA가 완성된후에 보다 습관적인 명세서가 작성되여야 한다. 이 문서는 부록 5에 있는 대부분의 자료들과 아주 류사하기때문에 이 실례연구에서는 생략한다. 구조화된 파라다임이 항공음식전문회사제품의 명세서에 리용되였을 때와 마찬가지로 (11.15) 이 시점에서 쏘프트웨어프로젝트관리계획이 작성된다. 부록 6은 소규모(3명) 쏘프트웨어개발기업체가 항공음식전문회사제품을 개발하기 위한 쏘프트웨어관리계획을 포함하고 있다. 이 계획은 IEEE SPMP형식에 준하고 있다.

12. 9. 객체지향분석단계에서의 난관

객체지향분석은 명세작성단계에 대한 한가지 특정한 방법이다. 그러므로 11.16에서 서술된 명세작성단계의 일반적난관들은 객체지향분석에 동등하게 적용된다. 특히 11.16에서 제시된 두번째 난관은 명세서(《무엇》)와 설계(《어떻게》)사이의 계선을 혼돈하기 쉽다는것이다. 이러한 위험성은 객체지향분석의 경우에 특별히 심하게 제기된다.

1.6에서 서술한바와 같이 객체지향분석으로부터 객체지향설계에로의 이행은 고전적파라다임작성에서 명세작성단계로부터 설계단계에로의 이행보다 훨씬 원활하다는것을 상기하자. 고전적파라다임작성에서 설계단계의 초기과제는 제품을 모듈들로 분해하는것이다. 반대로 클라스들 즉 객체지향설계단계의 《모듈들》은 객체지향분석단계에서 추출되여 객체지향설계단계에서의 세련을 위하여 준비된다. OOA단계에서 벌써 클라스들이 출현하는것은 OOA를 오래동안 리용하려는 유혹이 극히 강할수 있다는것을 의미한다.

실례로 클라스에로 방법들의 병합에 관한 론의를 고찰하자. 고전적명세작성단계의 한가지 과제는 목적하는 제품의 자료와 작용들을 결정하는것이다. 그러나 특정한 모듈에로 각이한 작용들의 병합은 설계단계까지 미루어야 한다. 왜냐하면 11.16에서 제시한바와 같이 먼저 제품은 전체로서 모듈로 어떻게 갈라 넣겠는가를 결정하여야 하기때문이다. 그러나 객체지향파라다임에서 이 과제는 분석단계에 속한다. 즉 객체지향분석단계에서 모듈(클라스)들과 그 속성들을 결정한다. 그리고 그 결과를 클라스선도로 묘사한다(12.4). 결국 명백히 클라스에로 방법들을 병합하기에 앞서 객체지향설계단계까지 기다려야 할 아무런 리유도 없다.

그러나 객체지향분석이 반복과정이라는것을 명심하는것이 중요하다. 각이한 모형들을 개선해 나가는 과정에 흔히 클라스모형의 많은 몫이 재조직되여야 한다. 클라스와 그 속성들을 재조직하는데는 시간이 많이 소비된다. 그다음 방법들을 재병합하는것은 불필요한 추가적인 재작업을 산생시킨다.

OOA과정의 매 걸음에서 반복기간에 재조직되여야 할 정보를 최소화하는것이 좋은 착상으로 된다. 그러므로 클라스에로 방법들의 병합은 객체지향분석단계에서 좀더 나갔으면 하는 생각이 있어도 설계단계까지 기다려야 한다.

요 약

객체지향분석이 소개되였다(12.1). 객체지향분석단계의 세개의 단계 즉 유스-케스모형화(12.3), 클라스모형화(12.4), 동 적모형화(12.5)가 서술되였다. 이 단계들 은 승강기문제(12.2 에서 재서술된)에 적용된다. 객체지향분석단계에서의 시험은 12.6 의 주제이다. 객체지향분석을 위한 CASE 도구들이 12.7 에서 서술된다. 이 장은 항공음식전문회사 실례연구(12.8)와 객체지향분석단계의 난관(12.9)으로 결속된다.

보 충

객체지향분석의 서로 다른 판본들을 서술한 초기의 문헌들로는 [Coad and Yourdon, 1991a; Rumbaugh et al., 1991; Shlaer and Mellor, 1992; and Booch, 1994]이 있다. 이 장에서 언급한바와 같이 이러한 기법들은 기본적으로 류사하다.

이러한 류형의 객체지향분석기법들 외에도 Fusion [Coleman et al., 1994]은 OMT [Rumbaugh et al., 1991]과 Objectory [Jacobson, Christerson, Jonsson, and Overgaard, 1992]를 비롯한 1세대기법들을 결합한 2세대OOA기법이다. 통합된 쏘프트웨어개발공정은 재콥숀, 브츠, 룸바우의 연구를 통합한것이다[Jacobson, Booch and Rumbaugh, 1999]. Catalysis는 또 하나의 중요한 객체지향방법론이다[D'Souza and Mills, 1999].

ROOM은 실시간쏘프트웨어를 위한 객체지향방법론이다[Selic, Gullekson, and Ward, 1995]. 실시간객체지향기술들에 대한 그이상의 정보는 문헌 [Awad, Kuusela, and Ziegler, 1996]에서 찾아 볼수 있다.

문헌 [Lee and Tepfenhart, 1997, and Fowler, 1997b]에서 UML1.0판에 대한 소개를 하였다. 문헌 [Booch, Rumbaugh, and Jacobson, 1999, and Rumbaugh, Jacobson, and Booch, 1999]에서 UML과 관련한 아주 상세한 내용들을 서술하였다. *Communications of the ACM* 1999년 10월호에는 UML의 리용과 관련한 아주 다양한 론문들이 있다.

이 장에서 리용한 후보클라스를 추출하는 명사추출기법은 문헌 [Juristo, Moreno, and López, 2000]. CRC카드는 문헌 [Beck and Cunningham, 1989]에서 처음으로 제기하였다. 문헌 [Wirfs-Brock, Wilkerson, and Wiener, 1990]은 CRC카드에 대한 아주 중요한 정보원천으로 되고 있다.

Communications of the ACM 1992년 9월호에는 객체지향분석에 관한 많은 기사들이 들어 있다. 객체지향분석기법에 대한 일부 비교에 대하여서는 문헌 [de Champeaux and Faure, 1992; Monarchi and Puhr, 1992; and Embley, Jackson, and Woodfield, 1995]을 비롯하여 여러가지 문헌들에 발표되였다. 객체지향과 구조화분석기법사이의 비교에 대하여서는 문헌 [Fichman and Kemerer, 1992]에서 고찰하였다.

객체지향프로젝트에서 반복의 관리는 문헌 [Williams, 1996]에서 서술하였다. 상태도(10.6.1)는 객체지향파라다임에로 확장되였다. 이에 대하여서는 문헌 [Coleman, Hayes, and Bear, 1992]와 [Harel and Gery, 1997]에서 서술하고 있다. 객체지향파라다임에서 설계명세서의 재리용은 문헌 [Bellinzona, Fugini, and Pernici, 1995]에서 서술하고 있다. 문헌 [Kazman, Abowd, Bass, and Clements, 1996]에서는 객체지향분석에 쓰이는 대본의 리용을

제기하고 있다.

문 제

12.1. 그림 12-9에 보여 준 다른 클라스에 대한 상태도를 개발함으로써 승강기문제 실례연구를 완성하시오.

12.2. 객체지향분석을 리용하여 문제 8.7의 자동화된 서고순환체계를 명시하시오.

12.3. 11.6의 유한상태기계형식화를 왜 객체지향분석에서 변화시키지 않고 리용할수 없는가?

12.4. 객체지향분석을 리용하여 문제 8.9의 ATM을 조종하기 위한 쏘프트웨어를 명시하시오. 여기서 카드읽기장치, 인쇄기, 현금분배기와 같은 하드웨어요소들의 세부들은 고찰할 필요가 없다. 그대신 단순히 ATM이 이 요소들에 지령을 보낼 때 그 지령들이 정확하게 실행된다는것을 가정하시오.

12.5. 제품에 불리한 영향을 주지 않으면서 클라스들이 도입될수 있는 객체지향분석과정에서의 최신측면은 무엇인가?

12.6. 클라스들이 의미있게 도입될수 있는 객체지향파라다임에서의 최초의 측면은 무엇인가?

12.7. 이 장에서 서술한 상태도와 다른 형식을 리용하여 동적모형을 표현할수 있는가를 설명하시오.

12.8. 왜 클라스들의 방법이 아니라 속성들이 객체지향분석기간에 결정된다고 생각하는가?

12.9. (과정안상 목표)객체지향분석을 리용하여 부록 1에 서술된 브로드랜즈지역 아동병원제품을 명시하시오.

12.10. (실례연구)클라스 **Meal**을 항공음식전문회사 실례연구(12.8)의 객체지향분석에 추가하시오.

12.11. (실례연구) 객체지향분석을 동적모형화로 시작하면 무슨 현상이 발생하는가를 결정하시오. 그림 12-6의 상태도로 시작하여 항공음식전문회사 실례연구에 대하여 객체지향분석과정을 완성하시오.

12.12. (실례연구) 11.4의 구조화체계분석을 12.8의 객체지향분석과 비교하고 대조하시오.

12.13. (쏘프트웨어공학독본) 교원은 문헌 [Harel and Gery, 1997]의 복제본을 배포할것이다. 상태도표를 OOA에서 상태도를 대신하여 응용할수 있는가?

참 고 문 헌

[Awad, Kuusela, and Ziegler, 1996] M. AWAD, J. KUUSELA, AND J. ZIEGLER, *Object-Oriented Technology for Real-Time Systems,* Prentice Hall, Upper Saddle River, NJ, 1996.

[Beck and Cunningham, 1989] K. BECK AND W. CUNNINGHAM, "A Laboratory for Teaching Object-Oriented Thinking," Proceedings of OOPSLA '89, *ACM SIGPLAN Notices* **24** (October 1989), pp. 1–6.

[Bellinzona, Fugini, and Pernici, 1995] R. BELLINZONA, M. G. FUGINI, AND B. PERNICI, "Reusing Specifications in OO Applications," *IEEE Software* **12** (March 1995), pp. 656–75.

[Booch, 1994] G. BOOCH, *Object-Oriented Analysis and Design with Applications,* 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994.

[Booch, Rumbaugh, and Jacobson, 1999] G. BOOCH, J. RUMBAUGH, AND I. JACOBSON, *The UML Users Guide,* Addison Wesley, Reading, MA, 1999.

[Coad and Yourdon, 1991a] P. COAD AND E. YOURDON, *Object-Oriented Analysis,* 2nd ed., Yourdon Press, Englewood Cliffs, NJ, 1991.

[Coleman, Hayes, and Bear, 1992] D. COLEMAN, F. HAYES, AND S. BEAR, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," *IEEE Transactions on Software Engineering* **18** (January 1992), pp. 9–18.

[Coleman et al., 1994] D. COLEMAN, P. ARNOLD, S. BODOFF, C. DOLLIN, H. GILCHRIST, F. HAYES, AND P. JEREMAES, *Object-Oriented Development: The Fusion Method,* Prentice Hall, Englewood Cliffs, NJ, 1994.

[D'Souza and Wills, 1999] D. F. D'SOUZA AND A. C. WILLS, *Objects, Components, and Frameworks with UML: The Catalysis Approach,* Addison-Wesley, Reading, MA, 1999.

[de Champeaux and Faure, 1992] D. DE CHAMPEAUX AND P. FAURE, "A Comparative Study of Object-Oriented Analysis Methods," *Journal of Object-Oriented Programming* **5** (March/April 1992), pp. 21–33.

[Embley, Jackson, and Woodfield, 1995] D. W. EMBLEY, R. B. JACKSON, AND S. N. WOODFIELD, "OO Systems Analysis: Is It or Isn't It?" *IEEE Software* **12** (July 1995), pp. 18–33.

[Fichman and Kemerer, 1992] R. G. FICHMAN AND C. F. KEMERER, "Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique," *IEEE Computer* **25** (October 1992), pp. 22–39.

[Fowler, 1997b] M. FOWLER WITH K. SCOTT, *UML Distilled,* Addison-Wesley, Reading, MA, 1997.

[Harel and Gery, 1997] D. HAREL AND E. GERY, "Executable Object Modeling with Statecharts," *IEEE Computer* **30** (July 1997), pp. 31–42.

[Jacobson, Booch, and Rumbaugh, 1999] J. RUMBAUGH, G. BOOCH, AND I. JACOBSON, *The Unified Software Development Process,* Addison-Wesley, Reading, MA, 1999.

[Jacobson, Christerson, Jonsson, and Overgaard, 1992] I. JACOBSON, M. CHRISTERSON, P. JONSSON, AND G. OVERGAARD, *Object-Oriented Software Engineering: A Use Case Driven Approach,* ACM Press, New York, 1992.

[Juristo, Moreno, and López, 2000] N. JURISTO, A. M. MORENO AND M. LÓPEZ, "How to Use Linguistic Instruments for Object-Oriented Analysis," *IEEE Software* **17** (May/June 2000), pp. 80–89.

[Kazman, Abowd, Bass, and Clements, 1996] R. KAZMAN, G. ABOWD, L. BASS, AND P. CLEMENTS, "Scenario-Based Analysis of Software Architecture," *IEEE Software* **13** (November/December 1996), pp. 47–55.

[Lee and Tepfenhart, 1997] R. C. LEE AND W. M. TEPFENHART, *UML: A Practical Guide to Object-Oriented Development,* Prentice Hall, Upper Saddle River, NJ, 1997.

[Monarchi and Puhr, 1992] D. E. MONARCHI AND G. I. PUHR, "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM* **35** (September 1992), pp. 35–47.

[Rumbaugh, Jacobson, and Booch, 1999] J. RUMBAUGH, I. JACOBSON, AND G. BOOCH, *The Unified Modeling Language Reference Manual,* Addison-Wesley, Reading, MA, 1999.

[Rumbaugh et al., 1991] J. RUMBAUGH, M. BLAHA, W. PREMERLANI, F. EDDY, AND W. LORENSEN, *Object-Oriented Modeling and Design,* Prentice Hall, Englewood Cliffs, NJ, 1991.

[Selic, Gullekson, and Ward, 1995] B. SELIC, G. GULLEKSON, AND P. T. WARD, *Real-Time Object-Oriented Modeling,* John Wiley and Sons, New York, 1995.

[Shlaer and Mellor, 1992] S. SHLAER AND S. MELLOR, *Object Lifecycles: Modeling the World in States,* Yourdon Press, Englewood Cliffs, NJ, 1992.

[USNO, 2000] "The 21st Century and the 3rd Millennium—When Will They Begin?" U.S. Naval Observatory, Astronomical Applications Department, aa.usno.navy.mil/AA/faq/docs/millennium.html, February 22, 2000.

[Williams, 1996] J. D. WILLIAMS, "Managing Iteration in OO Projects," *IEEE Computer* **29** (September 1996), pp. 39–43.

[Wirfs-Brock, Wilkerson, and Wiener, 1990] R. WIRFS-BROCK, B. WILKERSON, AND L. WIENER, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, NJ, 1990.

제1 3장. 설 계 단 계

지난 35년간 수백가지 설계기법들이 제시되였다. 일부는 현존기법들에 대한 변종이며 일부는 이전에 제안된 기법들과 근본적으로 다르다. 한두가지 설계기법들은 수만명의 쏘프트웨어공학자들에게 리용되였으며 대부분은 그 제안자들만이 리용하였다. 일부 설계전략들 특히 대학들에서 개발한것들은 확고한 리론적기초를 가지고 있다. 대학들에서 개발한것들을 포함하여 일부는 보다 실용주의적이다. 즉 그것들은 그 제안자들이 그 방법들이 실천에서 잘 동작한다는것을 알았기때문에 제시되였다. 대다수 설계기법들은 수동적이지만 자동화가 점차 설계의 중요한 측면으로 되고 있는데 다만 문서의 관리를 지원하기 위한 경우만 놓고 보아도 그렇다.

설계기법들이 이처럼 많음에도 불구하고 거기에는 일정한 기초적인 형식이 있다. 이 책의 주요화제는 어떤 제품의 두가지 본질적인 측면은 그것의 작용과 이 작용이 조작하는 자료이라는것이다. 그러므로 어떤 제품을 설계하는 두가지 기본적인 방법은 작용지향설계와 자료지향설계이다. 작용지향설계(*action-oriented design*)에서 중요한것은 작용들이다. 한가지 실례는 자료흐름분석인데 여기서 목적은 고도의 응집도를 가진 모듈들을 설계하는것이다(7.2). 자료지향설계(*data-oriented design*)에서는 자료가 우선적으로 고찰된다. 실례로 잭슨(Jackson)의 기법에서(13.5) 자료의 구조가 먼저 정의되고 그다음 그 자료구조에 따라 절차가 설계된다.

작용지향설계기법의 약점은 그것이 작용들에 집중하고 자료의 중요성은 2차로 한다는것이다. 자료지향설계기술은 이와 류사하게 자료를 우선 강조하며 작용을 덜 중요시한다. 해결대책은 객체지향기술을 리용하는것인데 이것은 작용과 자료에 같은 비중을 두고 있다. 이 장에서 작용지향 및 자료지향설계를 먼저 서술하고 그다음 객체지향설계가 서술된다. 하나의 객체가 작용과 자료를 모두 병합하고 있는것과 마찬가지로 객체지향설계는 작용지향설계와 자료지향설계의 특성을 결합시킨다. 그러므로 객체지향설계를 원만히 리해하기 위하여서는 작용지향설계 및 자료지향설계에 대한 기본적인 리해를 가지는것이 필요하다.

특정한 설계기법들을 시험하기에 앞서 설계단계에 관하여 일부 일반적인 주의를 줄 필요가 있다.

13. 1. 설계와 추상화

쏘프트웨어설계단계는 세개의 활동 즉 구성방식설계, 상세설계, 설계시험으로 이루어진다. 설계공정에 대한 연구는 명세서 즉 그 제품이 무엇을 하게 되는가에 대한 서술이다. 출력은 설계공정의 설계문서 즉 제품이 이것을 어떻게 달성하여야 하는가에 대한 서술이다.

구성방식설계(*architectural design*)(또는 일반설계, 론리설계, 고준위설계로 알려 져 있다.)기간에 제품의 모듈분해가 전개된다. 즉 명세서가 주의 깊게 분석되고 희망하는 기능을 가진 모듈구조가 생성된다. 이 활동의 결과는 모듈들의 목록과 모듈들이 어떻게 결합되게

되는가에 대한 서술이다. 추상화의 관점으로부터 구성방식설계기간에 일정한 모듈들의 존재가 가정되고 그다음 이 모듈들에 의하여 설계가 전개된다. 객체지향파라다임이 리용될때 1.6에서 설명한바와 같이 구성방식설계활동의 부분은 객체지향분석단계에서 수행된다(12장). 이것은 OOA에서의 첫 단계가 클라스들을 결정하는것이기때문이다. 하나의 클라스는 한가지 류형의 모듈이기때문에 모듈분해의 일부 과정은 분석단계에서 수행되였다.

다음활동은 상세설계(*detailed design*)인데 이것은 또한 모듈설계, 물리적설계, 저준위설계로 알려 져 있다. 이 단계에서 매 모듈이 세부적으로 설계된다. 실례로 특정한 알고리듬들이 선택되고 자료구조들이 선택된다. 또한 추상화의 관점으로부터 이 활동기간에 모듈들이 서로 련결되여 하나의 완전한 제품을 형성하게 된다는 사실이 무시된다.

이 두 단계의 과정은 제7장에서 서술한바와 같이 전형적인 추상화이다. 우선 최상위준위(전체 제품)가 아직 존재하지 않는 모듈들에 의하여 설계된다. 그다음 매 모듈들이 그것이 완전한 제품의 하나의 구성요소로 된다는것을 고려함이 없이 차례로 설계된다.

설계단계가 세개의 활동을 가지며 그 세번째 활동이 시험이라는것은 이미 서술하였다. 시험이 전체적인 쏘프트웨어개발 및 유지정비과정의 완전한 부분으로 되는것과 마찬가지로 그것이 설계의 완전한 부분이라는것을 강조하기 위하여 단어 단계(*stage*) 또는 걸음(*step*)이 아니라 활동(*activity*)이 리용되였다. 시험은 구성방식설계와 상세설계가 완성된 다음에만 수행되는 활동이 아니다.

여러가지 각이한 설계기법들을 이제 고찰한다. 먼저 작용지향기법을 설명하고 그다음 자료지향기법 그리고 마지막으로 객체지향기법들을 설명한다.

13. 2. 작용지향설계

7.2과 7.3에서는 어떤 제품을 고도의 응집도와 낮은 결합도를 가진 모듈로 분해하기 위한 한가지 리론적실례를 제시하였다. 이제 이러한 설계목적을 달성하기 위한 두가지 실천적기법들 즉 자료흐름분석(13.3)과 트랜잭션분석(13.4)을 서술한다. 리론적으로 자료흐름분석은 명세서들이 자료흐름도로 표현될 때마다 적용될수 있다. 그리고 (적어도 리론적으로) 매 제품은 DFD로 표현될수 있기때문에 자료흐름분석은 광범히 응용될수 있다. 그러나 실천적으로 많은 경우 보다 적당한 설계기법들이 존재하는데 특히 자료의 흐름이 다른 고찰보다 2차적인 제품들을 설계하는 경우에 그러하다.

다른 설계기법들이 필요한 실례들은 규칙에 기초한 체계(전문가체계), 자료기지, 트랜잭션처리제품들이다(13.4에서 서술된 트랜잭션분석은 트랜잭션처리제품들을 모듈로 분해하기 위한 좋은 방법이다.).

13. 3. 자료흐름분석

자료흐름분석(*Data Flow Analysis*; DFA)은 고도의 응집도를 가진 모듈들을 얻기 위한 설계기법이다. DFA는 대부분의 명세작성기법과 함께 리용될수 있다. 이 절에서 DFA는 구조화체계분석(11.3)과 함께 제시된다. 이 기법에 대한 입력은 자료흐름도이다. 중요한 점은 일단 DFD가 완성되면 쏘프트웨어설계자는 그 제품에 대한 입력과 그 제품의 출력

을 고려하여 정확하고 완전한 정보를 가지게 된다는것이다.

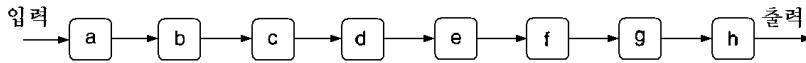



그림 13-1. 자료의 흐름과 제품의 작용을 보여 주는 자료흐름도

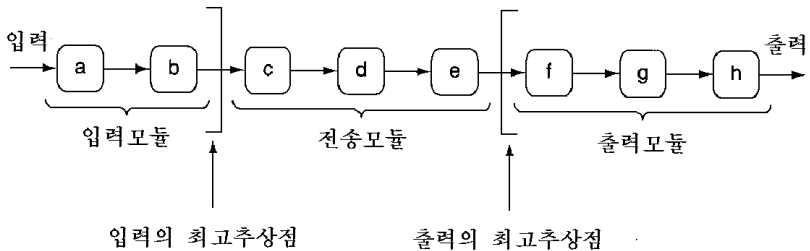



그림 13-2. 입력과 출력의 최고추상점들

그림 13-1의 DFD에 의하여 표현된 제품에서의 자료흐름을 고찰하자. 이 제품은 어쨌든 입력을 출력으로 변환한다. DFD의 어떤 점에서 입력은 입력이기를 그만 두고 어떤 종류의 내부자료로 된다. 그러면 어떤 이후점에서 이 내부자료들은 출력의 자격을 가지게 된다. 이것은 그림 13-2에서 보다 자세히 보여 준다. 그 입력이 입력으로 될 자격을 잃고 단순히 제품이 조작하는 내부자료로 되는 점을 입력의 최고추상점(*point of hightest abstraction of input*)이라고 부른다. 이와 류사하게 출력의 최고추상점(*point of hightest abstraction of output*)은 출력이 어떤 종류의 내부자료로서가 아니라 출력으로서 식별될수 있는 자료흐름에서의 첫 점이다.

입력과 출력의 최고추상점을 리용하여 제품을 세개의 모듈 즉 입력모듈, 변환모듈, 출력모듈로 분해한다. 이제 매 모듈이 차례로 취해 지고 그것의 최고추상점이 찾아 지며 모듈은 다시 분해된다. 이 절차는 매 모듈이 단일한 작용을 수행할 때까지 계단식으로 계속된다. 즉 설계는 고도의 응집도를 가진 모듈들로 이루어 진다. 결국 그토록 많은 다른 쏘프트웨어공학기법들의 기초로 되는 계단식세련은 또한 자료흐름분석의 기초를 이룬다.

공정하게 말하면 최저결합가능성을 달성하기 위하여 분해에 약간한 변경이 가해 져야 할수도 있다. 자료흐름분석은 고도의 응집도를 달성하기 위한 한가지 방법이다. 혼합식구조화설계의 목적은 고도의 응집도뿐아니라 낮은 결합도를 달성하는것이다. 낮은 결합도를 달성하기 위하여서는 때때로 설계에서 약간한 변경을 가하는것이 필요하다. 실례로 DFA 는 결합도를 고려하고 있지 않기때문에 DFA 를 리용하여 구성된 설계에서 무심결에 결합도조종이 일어 날수도 있다. 이와 같은 경우에 필요한것은 조종이 아니라 자료가 통과되도록 포함된 두개의 모듈을 변경하는것이다.

13. 3. 1. 자료흐름분석실례

UNIX wc 편의프로그람과 류사하게 어떤 파일이름을 입력으로 하고 그 파일안의 단어수를 귀환하는 제품을 설계하는 문제를 고찰하자.

그림 13-3은 자료흐름도를 묘사하고 있다. 거기에는 5개의 모듈이 있다. 모듈 read file name은 파일의 이름을 읽으며 그다음 그것은 모듈 validate file name에 의하여 확인된다. 확인된 이름은 모듈 count number of words에로 넘겨 지며 이 모듈은 단어개수를 정확히 계산한다. 총 단어수는 모듈 format word count로 넘겨 지고 최종적으로 형식화된 총 단어수가 출력을 위하여 모듈 display word count에로 넘겨 진다.

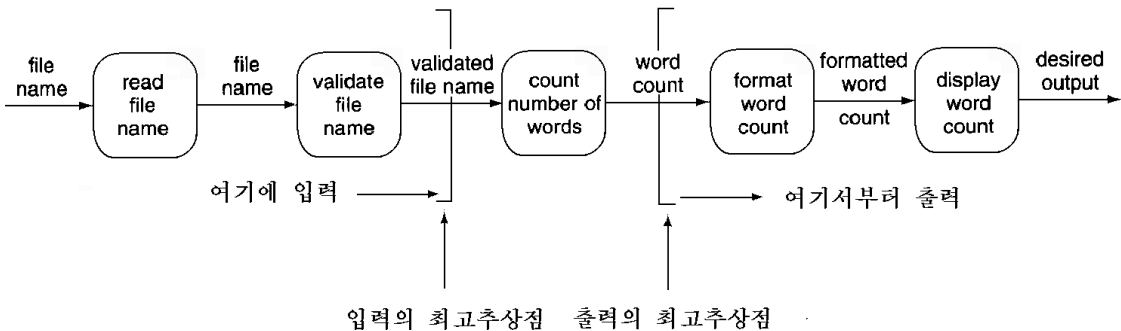



그림 13-3. 자료흐름도: 첫번째 세련

자료흐름을 검토할 때 초기입력은 파일이름(*file name*)이다. 이것이 확인된 파일이름(*validated file name*)으로 될 때 그것은 여전히 하나의 파일이름이며 그러므로 입력자료로서의 자격을 잃지 않는다. 그러나 모듈 count number of words를 고찰하면 그것의 입력은 validated file name이며 출력은 word count이다. 이 모듈의 출력은 자격에서 제품에 대한 입력과 완전히 다르다. 입력의 최고추상점은 그림 13-3에 지적된것과 같다. 류사하게 모듈 count number of words로부터의 출력에 어떤 종류의 형식화를 진행한다고 하여도 그것은 본질상 모듈 count number of words로부터 나올 때의 출력이다. 그러므로 출력의 최고추상점은 그림 13-3에 보여 준것과 같다.

이 두개의 최고추상점을 리용하여 제품을 분해한 결과를 그림 13-4의 구조도표에 보여 주었다. 그림 13-4는 또한 그림 13-3의 자료흐름도가 어느 정도 단순하다는것을 보여준다. DFD는 만일 사용자가 명시한 파일이 존재하지 않으면 무슨 현상이 발생하는가에 대응하는 론리적흐름을 보여 주지 않는다. 모듈 read and validate file name은 하나의 상태기발(*status flag*)을 모듈 perform word count에 귀환하여야 한다. 만일 이름이 타당하지 않으면 그것은 모듈 perform word count에 의하여 무시되며 어떤 종류의 오유통보가 인쇄된다. 그러나 만일 이름이 타당하면 그것은 모듈 count number of words에 넘겨 진다.

일반적으로 조건적자료흐름이 존재하는 곳마다 대응하는 조종흐름이 요구된다.

그림 13-4에서 두개의 모듈 read and validate file name과 format and display word count는 통신적인 응집도를 가진다(7.2.5) 이 모듈들은 더 분해되여야 한다. 최종결과를 그림 13-5에 보여 주었다. 8개의 모듈전부가 자료결합을 가지든가 또는 그것들사이의 결

합이 없이 기능적인 응집도를 가진다.

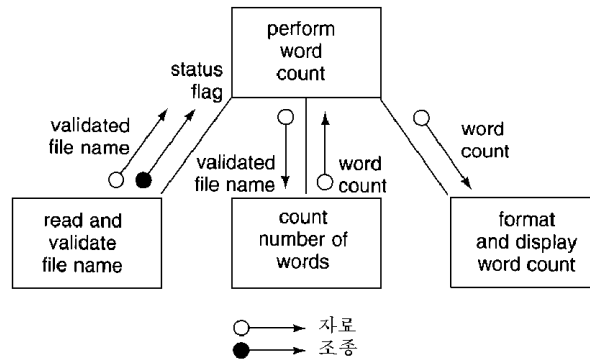



그림 13-4. 구조도: 첫번째 세련

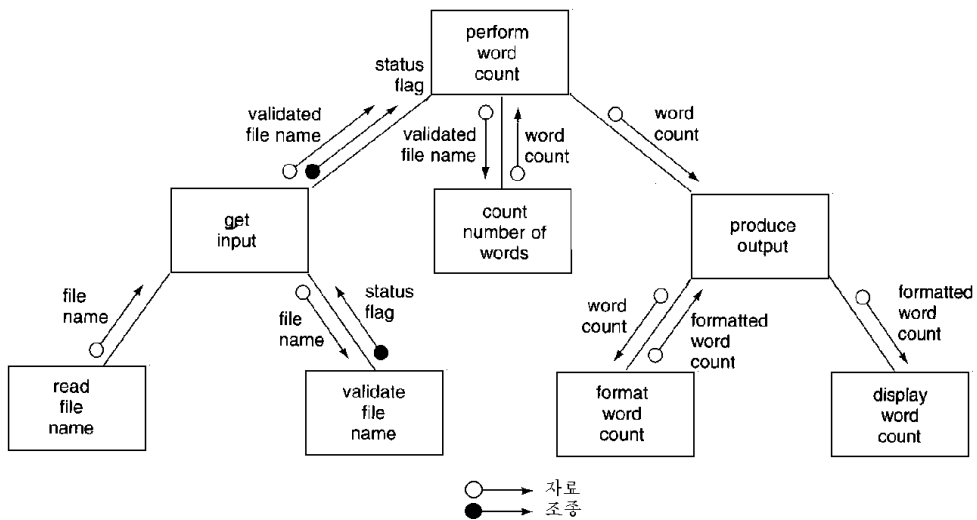



그림 13-5. 구조도: 두번째 세련

모듈이름	read file name
모듈형	함수
귀환값형	String
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	사용자는 지령문자렬 word count <file name>으로 제품을 호출한다. 조작체계호출을 리용하여 이 모듈은 사용자가 입력한 지령문자렬의 내용을 호출하고 <file name>을 추출하고 모듈의 값으로 그것을 귀환한다.

모듈이름	validate file name
모듈형	함수
귀환값형	Boolean
입력변수	file name : string
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	이 모듈은 조작체계를 호출하여 파일 file name이 존재하는가를 결정한다. 모듈은 파일이 존재하면 **true**, 그렇지 않으면 **false**를 귀환한다.

모듈이름	count number of words
모듈형	함수
귀환값형	Integer
입력변수	validated file name : string
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	이 모듈은 validated file name이 여러 행들로 분할된 본문파일인가를 결정한다. 그렇다면 모듈은 본문파일에 있는 단어의 수를 귀환하고 그렇지 않으면 −1을 귀환한다.

모듈이름	product output
모듈형	함수
귀환값형	**void**
입력변수	word count : integer
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	format word count 변수: word count : string formatted word count : string display word count 변수: formatted word count : string
해설	이 모듈은 호출모듈에 의하여 넘겨 진 옹근수 word count를 가지고 있으며 설계명세에 따라 형식화된 옹근수를 얻을수 있도록 format word count를 호출한다. 그다음 display word count를 호출하여 인쇄된 행들을 얻어 낸다.

그림 13-6. 실례의 네개 모듈에 대한 상세설계

```
void perform word count()
{
  String    validated file name;
  int       word count;
  if (get input (validated file name) is false)
    print "error 1 : file does not exit";
  else
  {
    word count를 count number of words (validated file name)와 같게 설정한다;
    if (word count is equal to −1)
      print "error 2 :  file is not a text file"
    else
      produce output (word count);
  }
}

Boolean get input (String validated file name)
{
  String     file name;
  file name = read file name ();
  if  (validate file name (file name) is true)(
  {
    validated file name을 file name으로 설정한다;
    return false;
  }
  else
    return false;
}

void display word count (String formatted word count)
{
  print formatted word count ,  left justified;
}

String format word count ( int word count);
{
  return "File contains"  word count "words");
}
```

그림 13-7. 실례의 네가지 방법에 대한 상세설계의 PDL(의사코드)표현

여기까지에서 구성방식설계가 완성되였으며 그 다음단계는 상세설계이다. 여기서 자료구조들이 선택되고 알고리듬들이 선택된다. 그다음 매 모듈에 대한 상세설계가 실현을

위하여 프로그람작성자에게 넘겨 진다. 사실상 쏘프트웨어제품생산의 다른 매 단계들과 마찬가지로 시간제약은 쏘프트웨어실현이 모든 모듈들을 코드화할 책임이 있는 개별적프로그람작성자에 의해서가 아니라 하나의 개발팀에 의하여 이루어 질것을 요구한다. 이러한 원인으로 하여 매 모듈들의 상세설계는 다른 임의의 모듈을 참고하지 않고 리용될수 있도록 제시되여야 한다. 8개의 모듈중에서 4개에 대한 상세설계가 그림 13-6에 제시되였다. 다른 4개의 모듈들은 다른 형식으로서 제시된다.

그림 13-6의 설계는 프로그람작성언어와는 무관계하다. 그러나 만일 관리자측이 상세설계를 시작하기전에 실천언어를 결정하기로 한다면 상세설계를 표현하기 위하여 프로그람서술언어(*Program Description Language*; PDL)를 리용하는것이 한가지 흥미 있는 대안으로 된다(의사코드(*pseudocode*)는 PDL에 대한 이전의 이름이다.). PDL은 본질상 선택된 실현언어의 조종명령문(*control statement*)으로 련결된 설명문들로 이루어 진다. 그림 13-7은 C++ 또는 Java 양상의 PDL로 작성한 이 제품의 나머지 4개 모듈에 대한 상세설계를 보여 주고 있다. PDL은 그것이 일반적으로 명백하고 간결하며 따라서 실현단계는 보통 설명문들을 련관된 프로그람작성언어로 순수 번역하여 이루어 진다는 우월성을 가지고 있다. 부족점은 때때로 설계자들이 지나친 세부에 들어 가서 PDL상세설계가 아니라 모듈의 완전한 코드실현을 초래하는 경향이 존재한다는것이다.

상세설계는 완전히 문서화되고 성과적으로 시험된후에 코드작성을 위하여 실현팀에 넘겨 진다. 그다음 제품은 쏘프트웨어개발공정의 나머지 단계들을 거치게 된다.

13. 3. 2. 확장

독자들은 이 실례가 약간 인공적이라는것을 느낄수 있는데 실례에서 자료흐름도(그림 13-3)는 다만 하나의 입력흐름과 하나의 출력흐름을 가지고 있다. 보다 복잡한 상황에서 무슨 현상이 발생하는가를 보기 위하여 그림 13-8을 고찰하자. 여기에는 4개의 입력흐름과 5개의 출력흐름이 있는데 상황은 현실에 보다 가깝게 대응하고 있다.

그림 13-8. 다입출력흐름을 가진 자료흐름도

다입출력흐름이 존재할 때 진행해 나가는 방법은 매 입력흐름에 대한 입력의 최고추

상점과 매 출력흐름에 대한 출력의 최고추상점을 찾는것이다. 그다음 이 점들을 리용하여 주어 진 자료흐름도를 원래보다 적은 입출력흐름을 가진 모듈들로 분해한다. 이 방법은 매 최종모듈들이 고도의 응집도를 가질 때까지 계속 진행해 나간다. 마지막으로 매 모듈쌍들사이의 결합도를 결정하고 임의의 필요한 조절을 진행한다.

자료흐름분석과정은 아래의 란에 종합한다.

자료흐름분석을 진행하는 방법

다음의 세 단계는 최종모듈들이 고도의 응집도를 가질 때까지 반복한다.

・ 매 입력흐름에 대한 입력의 최고추상점을 찾는다.
・ 매 출력흐름에 대한 출력의 최고추상점을 찾는다.
・ 이 최고추상점들을 리용하여 자료흐름도를 분해한다.

마지막으로

・ 만일 최종결합도가 너무 세면 설계를 조절한다.

13. 4. 트랜잭션분석

트랜잭션(*transaction*)이란 《어떤 요청을 처리한다.》 또는 《오늘 주문자의 목록을 인쇄한다.》와 같은 제품사용자의 관점으로부터의 어떤 조작을 말한다. 자료흐름부분은 거래업무처리형제품들에서는 적당치 않다. 이러한 제품들에서는 륜곽적으로는 류사하지만 세부적으로는 차이나는 많은 련관된 작용들이 수행되여야 한다. 한가지 전형적인 실례는 자동출납기를 조종하는 쏘프트웨어이다.

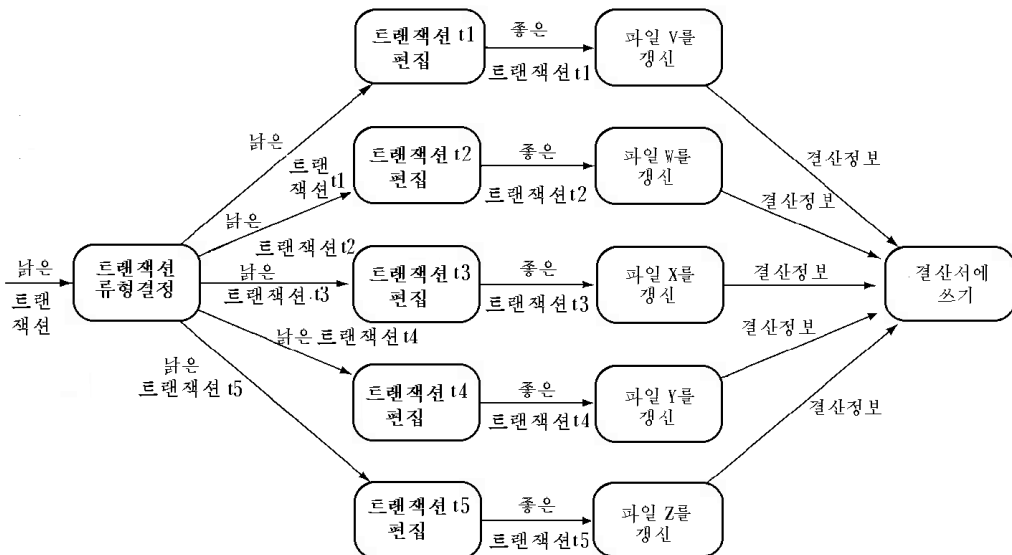



그림 13-9. 전형적인 트랜잭션처리체계

손님이 자화카드를 구멍에 넣고 통과암호를 입력하면 당좌예금, 시좌예금 또는 신용카드예금을 하는것, 구좌에서의 자금대출, 구좌에서 잔고결정과 같은 작용들을 수행한다. 이러한 류형의 제품을 그림 13-9에 보여 주었다. 이와 같은 제품을 설계하기 위한 한가지 좋은 방법은 그것을 두 부분 즉 분석기와 처리기로 가르는것이다. 분석기는 트랜잭션류형을 결정하고 이 정보를 처리기에 보내며 처리기는 트랜잭션을 수행한다.

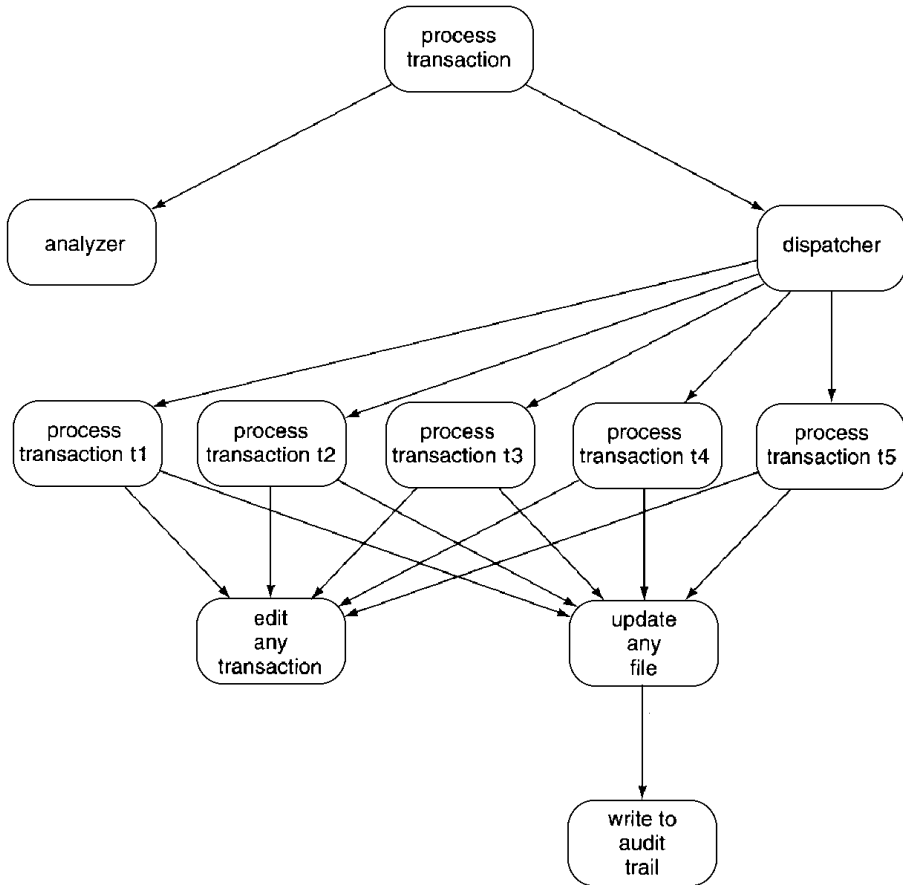



그림 13-10. 전형적인 트랜잭션처리체계의 대략적인 설계

이러한 류형의 한가지 불충분한 설계를 그림 13-10에 보여 주는데 그것은 론리적인 응집도를 가진 두개의 모듈 edit any transaction과 update any flie을 가지고 있다. 한편 5개의 매우 류사한 편집모듈들과 5개의 매우 류사한 갱신모듈들을 작성하는것은 로력랑비인것 같다. 이에 대한 해결방안은 쏘프트웨어의 재리용이다(8.1). 즉 하나의 기본편집모듈이 설계되고 코드작성되고 문서화되고 시험되며 그다음 그것이 5번 실례생성된다. 매 판본들은 약간 차이나지만 그 차이는 이 방법의 유용성을 충분히 담보할만큼 작다. 이와 류사하게 하나의 기본갱신모듈이 5번 실례생성되여 5개의 각이한 갱신모듈들을 만족시키도록 약간 변경될수 있다. 최종설계는 고도의 응집도와 낮은 결합도를 가지게 된다.

트랜잭션분석방법을 아래의 란에 종합한다.

트랜잭션분석을 진행하는 방법

· 구조는 두개의 요소 즉 분석기와 처리기로 구성된다.
· 매개의 련관된 작용들의 모임에 대하여 하나의 기본모듈을 설계하고 그것을 필요한 회수만큼 실례생성한다.

13. 5. 자료지향설계

자료지향설계의 리면에 있는 기본원리는 조작하려는 자료의 구조에 따라 제품을 설계하는것이다. 다시 말하여 우선 자료의 구조가 결정되고 그다음 매 절차들이 조작하려는 자료와 같은 구조로 주어 진다. 이와 같은 류형의 자료지향기법들은 많이 존재하고 있다. 가장 널리 알려 진것들로서는 미카엘 잭슨(Michael Jackson)[Jackson, 1975], 와니어(Warnier) [Warnier, 1976], 오르(Orr)[Orr, 1981]의 기법들인데 이 세가지 기법들은 많은 류사성을 가지고 있다.

자료지향설계는 작용지향설계처럼 그렇게 보편적이지는 못하며 객체지향파라다임의 출현으로 하여 크게 류행되지 않고 있다. 지면상의 리유로 하여 이 책에서는 자료지향설계를 더이상 론의하지 않는데 흥미 있는 독자들은 우에 제시된 참고문헌들을 보시오.

13. 6. 객체지향설계

객체지향설계(*object-oriented design*; OOD)의 목적은 객체 즉 객체지향분석단계에서 추출된 클라스 및 부분클라스들의 실례에 의하여 제품을 설계하는것이다. C, COBOL, FORTRAN, Pascal과 같은 고전적인 언어들은 이와 같은 객체들을 지원하지 않는다. 이것은 OOD가 Smalltalk[Goldberg and Robson, 1989], C++[Stroustrup, 1991], Eiffel[Meyer, 1992b], Ada95[ISO/IEC 8652, 1995], Java[Flanagan and Loukides, 1997]와 같은 객체지향언어들이 사용자들에게 접근가능하다는것을 의미하는것 같을수도 있다.

실정은 그렇지 않다. 비록 OOD가 고전적인 언어들로 지원되지는 않는다 하여도 OOD의 많은 부분들은 리용될수 있다. 7.7에서 설명한것처럼 클라스는 계승을 가지는 추상자료류형이며 객체는 클라스의 실례이다. 계승을 지원하지 않는 어떤 실현언어를 리용할때의 해결방안은 그 프로젝트에서 리용된 프로그람작성언어로 달성될수 있는 OOD의 측면들을 리용하는것 즉 추상자료류형설계(*abstract data type design*)를 리용하는것이다. 추상자료류형은 사실 형(**type**)명령을 지원하는 임의의 언어로 실현될수 있다. COBOL과 같은 고전적언어에서 조차도 자료교갑화를 실현하는것은 여전히 가능할수 있다. 그림 7-29는 모듈로서 시작하고 객체로서 끝나는 설계개념의 계층구조를 보여 주고 있다. 완전한 OOD가 가능하지 않은 경우들에 개발자들은 자기들의 설계가 설계언어가 지원하는 그림 7-29의 계층구조에서 최상의 가능한 개념을 리용한다는것을 확증하기 위하여 노력하여야

한다.

OOD는 다음의 네 단계로 이루어 진다.

1. 매 대본에 대하여 호상작용도를 구성한다.
2. 세부클라스도를 작성한다.
3. 객체들의 의뢰자에 의하여 제품을 설계한다.
4. 상세설계를 진행한다.

이제 OOD를 한가지 실례로서 례증한다. 앞에서와 마찬가지로 단순하게 하기 위하여 한대의 승강기를 가진 승강기문제가 제시된다. 같은 실례를 리용함으로써 문제 그자체의 작은 세부들에 대하여 념려하지 않고 각이한 방법들을 비교할수 있다.

13.7. 승강기문제: 객체지향설계

1단계. 매 대본에 대한 호상작용도를 구성한다 UML은 두가지 류형의 호상작용도 즉 순차도와 협동도를 지원한다. 이 두개의 선도들은 동일한것들 즉 객체들과 그 객체들을 통과하는 통보들을 보여 준다. 그러나 이 두 선도에서 중점은 서로 다르다. 순차도는 통보들의 명백한 시간적인 순서를 강조하고 있다. 그러므로 순차도는 사건발생의 순서가 중요한 정황에서 쓸모 있다. 협동도는 객체들사이의 관계를 강조하고 있으며 쏘프트웨어 제품의 구조를 리해하기 위한 하나의 강력한 수단으로 된다.

그림 12-10의 대본(여기서는 그림 13-11로 재현되고 있다.)을 고찰하자. 그림 13-12의 대응하는 순차도에서 사용자들과 객체들은 수직선들로 표현된다. UML에서 어떤 클라스의 실례(하나의 객체)는 밑선을 친 소문자로 된 클라스이름으로 표현된다.

시간은 상단에서부터 하단으로 향한다. 그림 13-11의 대본에서 첫번째 사건은 사용자 A가 상승층단추를 누른다는것이다. 이것은 **1. press floor button**으로 표식한 User A로부터 **floor button**에로 향하는 수평선으로 표현된다. 다음으로 층단추는 승강기조종기에 층단추가 눌러 졌다는것(대본의 사건 2)을 통보한다. 이것은 **floor button**로부터 **elevator controller**로 향하는 수평선으로 표현된다. 승강기조종기는 그다음 련관된 층 단추에 통보를 보내여 그 단추를 켠다(대본의 사건 3). 이것은 3. turn button on으로 표식한 **elevator controller** 로부터 **floor button**에로 향하는 수평선으로 표현된다. 대본에서 4번째 사건은 승강기조종기가 일련의 통보들을 승강기에 보낸 결과로 승강기가 3층에 도착한다는것이다. 이것은 한층 우로 올라 가기 위하여 **elevator controller**가 **elevator**에 보내는 통보를 반복하는것으로서 모형화된다. 이 반복은 4. *move up one floor로 지적된다. 순차도의 나머지부분은 류사하다.

그림 13-12는 그림 13-11의 대본안에 있는 작용자들 즉 대본의 사건들에서 능동적인 사용자들과 객체들을 먼저 렬거하는것으로서 구성되였다. 작용자들은 User A(사건 1과 8), **floor button**(사건 1, 2, 3, 5), **elevator controller**(사건 2부터 7까지와 사건 9부터 7까지), **elevator**(사건4, 12, 16), **elevator doors**(사건2, 10, 14, 16), **elevator**(사건 9, 10, 13)들이다.

1. 사용자 A는 3층에서 상승층 단추를 눌러 승강기를 요청한다.
   사용자 A는 7층으로 가려고 한다.
2. 층 단추는 승강기조종기에서 층 단추가 눌러 졌다는것을 통보한다.
3. 승강기조종기는 상승층 단추가 켜지도록 상승층 단추에 통보를 보낸다.
4. 승강기조종기는 승강기가 3층으로 이동하도록 승강기에 일련의 통보들을 보낸다. 승강기에는 사용자 B가 있는데 그는 1층에서 승강기에 올라 9층 승강기단추를 눌렀다.
5. 승강기조종기는 상승층 단추가 꺼지도록 상승층 단추에 통보를 보낸다.
6. 승강기조종기는 승강기문을 열도록 승강기문에 통보를 보낸다.
7. 승강기조종기는 시간계수기를 동작시킨다.
   사용자 A가 승강기에 오른다.
8. 사용자 A는 7층 승강기단추를 누른다.
9. 승강기단추는 승강기조종기에 승강기단추가 눌러 졌다는것을 통보한다.
10. 승강기조종기는 7층 승강기단추가 켜지도록 그 단추에 통보를 보낸다.
11. 승강기조종기는 설정시간이 지나면 승강기문을 닫도록 승강기문에 통보를 보낸다.
12. 승강기조종기는 승강기가 7층으로 이동하도록 승강기에 일련의 통보들을 보낸다.
13. 승강기조종기는 7층 승강기단추가 꺼지도록 그 단추에 통보를 보낸다.
14. 승강기조종기는 사용자 A가 승강기에서 내리도록 승강기문을 열기 위하여 승강기문에 통보를 보낸다.
15. 승강기조종기는 시간계수기를 통작시킨다.
    사용자 A는 승강기에서 내린다.
16. 승강기조종기는 설정시간이 지나면 승강기문을 닫도록 승강기문에 통보를 보낸다.
17. 승강기조종기는 사용자 B를 태우고 승강기가 9층으로 이동하도록 승강기에 일련의 통보들을 보낸다.

그림 13-11. 표준대본의 두번째 반복

이 작용자들을 표현하는 통들이 그림 13-12의 상단에 그려 지고 그다음 2중수직선이 그려 진다. 그림 13-11의 대본안에 있는 매 사건이 차례로 그림 13-12의 순차도에 삽입된다. 사건 1에서 User A는 층단추를 누른다. 이것을 모형화하기 위하여 하나의 수평선을 User A(작용을 일으킨다.)로부터 floor button(수신자)에로 그린다. 사건 2에서 7층 상승층 단추는 승강기조종기에 그 층단추가 눌러 졌다는것을 통보한다. 그러므로 **floor button**(작용을 일으킨다.)으로부터 **elevator controller**(수신자)에로 하나의 수평선이 그려 진다. 다른 수평선들도 이와 류사하게 그려 진다.

그림 13-12는 두개의 순환을 가지고 있는데 그것들은 승강기조종기내에서 시간계수기의 작용들을 표현하고 있다. 첫번째 순환은 7. start timer로 표식되였다. 그림 13-12의 대본의 사건 7에 서술된것처럼 시간계수기는 문이 문을 열라고 통보를 보낸후에 동작을

시작한다(보다 정확하게는 시간계수기는 문이 승강기조종기에 문을 열었다는것을 통보하였을 때 동작을 시작한다. 이러한 실례의 많은 서술은 간단하게 하기 위하여 생략하였다.).

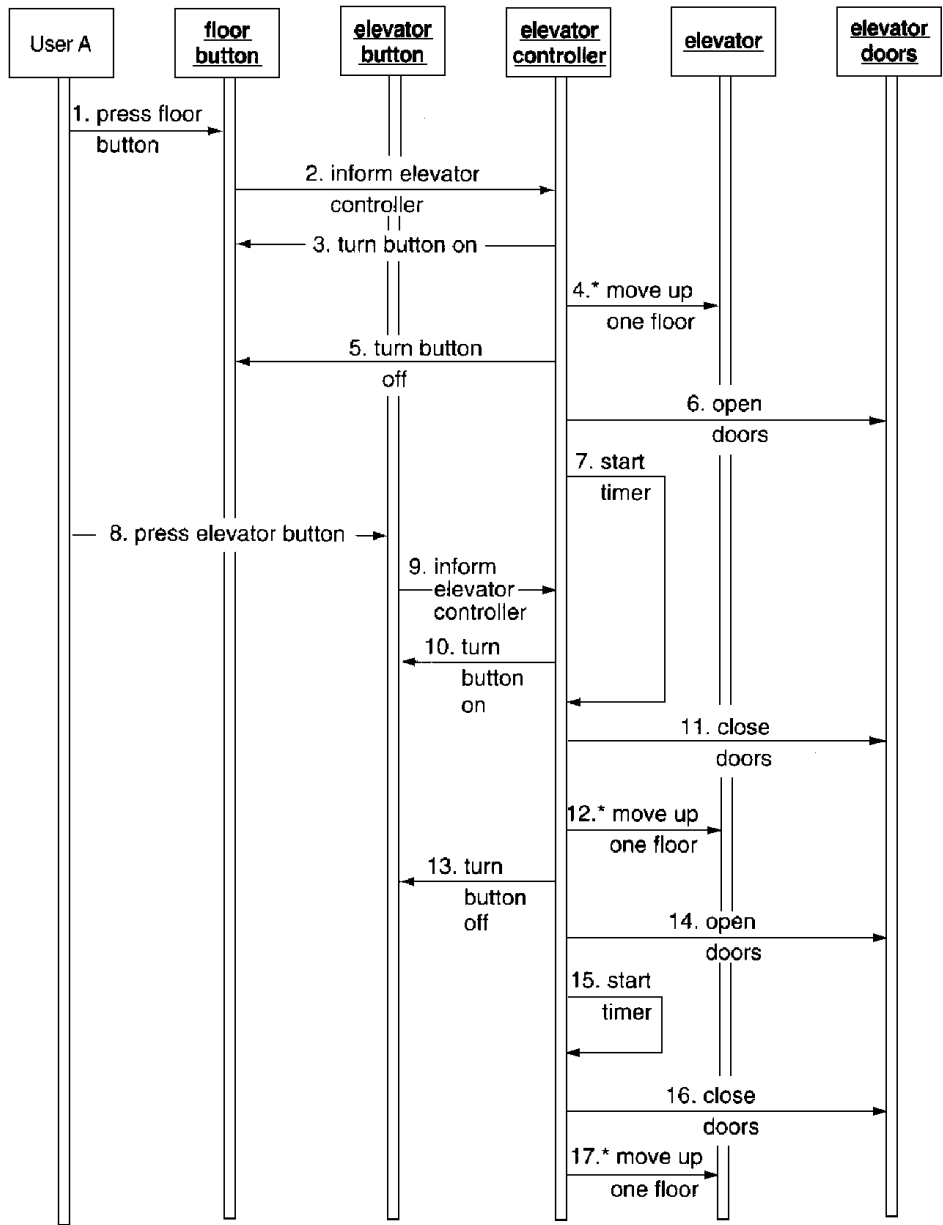



그림 13-12. 그림 13-11의 대본을 위한 순차도

문이 열릴 때 사용자 A는 7층 승강기단추를 누르며(사건 8) 승강기단추는 승강기조종기에 이에 대하여 통보한다(사건 9). 그리고 승강기조종기는 통보를 승강기단추에 보내

여 조명을 켠다(사건 10). 그다음 설정시간초과가 발생하며(사건 11) 승강기조종기는 문에 통보를 보내여 문을 닫는다. 이것은 그림 13-12의 첫번째 시간계수순환의 끝이 사건 10과 사건 11사이에 있는 **elevator controller**를 표현하는 2중수직선과 마주치게 되는 리유이다. 같은 대본에 대한 협동도를 그림 13-13에 보여 준다.

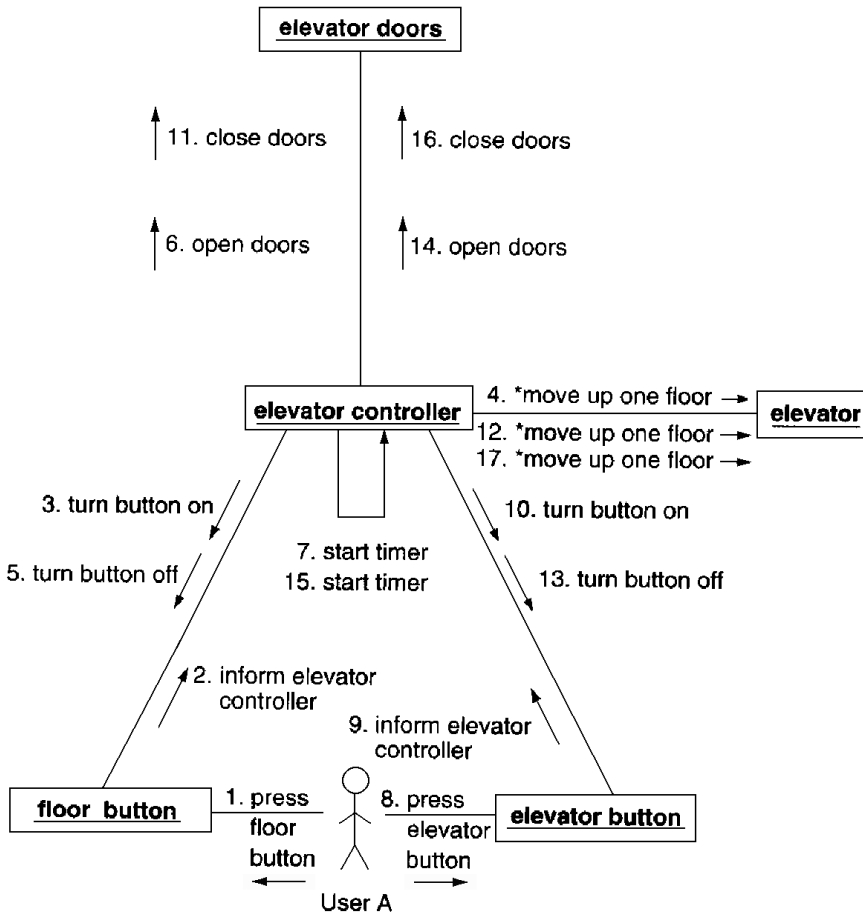



그림 13-13. 그림 13-11의 대본을 위한 협동도

순차도와는 달리 협동도는 **elevator controller**가 노는 중심적인 역할을 명확히 보여준다. 사실 대본에서 두개를 제외한 모든 통보들은 **elevator controller**에로 또는 그로부터 보내 진다. 비록 두 선도가 모두 정확하게 같은 정보를 포함하고 있다고 하여도 그림 13-12의 순차도의 시간정보는 그림 13-13의 협동도에서 명백하지 않다. 경험은 매 프로젝트에서 설계팀이 두개의 호상작용선도가운데서 어느것이 더 적당한가를 결정하여야 한다는것이다. 많은 경우에 두개가 다 필요할것이다. 사실 쏘프트웨어가 만일 OOA와 OOD를 위한 CASE도구를 리용하여 개발된다면 그 모두는 요구에 따라 어느 한 선도를 발생

할수 있다. 이것은 팀이 즉흥적으로 적당한 호상작용선도를 선택할수 있게 한다.

그림 13-13의 협동도는 이 대본에 관하여 그림 13-12의 협동도와 같은 방식으로 구성되였다. 즉 작용자들은 그림 13-11의 대본으로부터 추출되고 그림 13-12에 그려 졌다. 그다음 매 사건들은 차례로 조사되여 이 선도에 포함되였다. 실례로 사건 3에서 승강기조종기는 승강기단추에 조명을 켤것을 지시한다. 그러므로 **elevator controller**로부터 **floor button**에로 선이 그어 지며 그 선과 평행으로 작은 화살표가 배치되고 3. turn button on으로 주석이 달아 진다. 다른 사건들도 이와 류사하게 삽입된다.

그림 13-12의 순차도와 그림 13-13의 협동도를 구성하는 공정에서의 유일한 차이는 순차도에서 사건들은 련관된 작용자들의 이름을 포함하고 있는 통으로 시작되는 2중수직선에 련결되는 수평선들로 표현되며 반면에 협동도에서는 사건들이 작용자들의 이름을 포함하고 있는 통들을 련결하는 선들을 따라 배치된 주석 붙은 화살표들로 표현된다는것이다.

2단계. 세부클라스도를 구성한다 OOA단계의 클라스도는(12.5) 클라스들과 그 속성들은 묘사하고 있지만 그것들의 작용자는 복사하지 않고 있다. 방법들은 OOD단계의 세부클라스도에 삽입된다.

제품의 모든 작용들을 결정하는것은 매 대본의 호상작용선도들을 조사함으로써 수행된다. 이것은 간단하다. 어려운것은 어느 작용들이 매개 클라스와 관련되여야 하는가를 확정하는 방법을 결정하는것이다.

하나의 작용은 하나의 클라스들과 그 클라스의 하나의 객체에 통보를 보내는 하나의 의뢰자에게 할당될수 있다(어떤 객체의 한 의뢰자는 그 객체에 통보를 보내는 하나의 프로그람단위이다.). 이런 작용을 어떻게 할당하는가를 결정하기 위한 한가지 방도는 정보은폐에 기초하는것이다. 즉 어떤 클라스의 상태변수들은 **private**(그 클라스의 한 객체내에서만 접근할수 있다.)든가 **protected**(그 클라스의 한 객체 또는 그 클라스의 한 부분클라스내에서만 호출할수 있다.)로 선언되여야 한다. 따라서 상태변수에 대하여 수행된 작용들은 그 클라스에 대하여 국부적이여야 한다.

정보은폐가 전혀 요구되지 않는다고 하여도 만일 어떤 특별한 작용이 어떤 객체에 대한 서로 다른 여러 의뢰자에 의하여 호출된다면 그 객체의 매개 의뢰자들안에서 복사를 하는것이 아니라 그 객체의 하나의 방법으로써 실현된 그 작용의 단일한 복사를 하는것이 합리적이다. 작용을 어디에 배치하는가를 결정하는 세번째 방법은 책임구동설계방법을 리용하는것이다.

7.6에서 설명한바와 같이 객체지향파라다임의 한가지 중요한 측면은 책임구동설계원리이다. 만일 어떤 의뢰자가 하나의 객체에 어떤 통보를 보내면 그 객체는 의뢰자의 요청을 수행할 책임을 지게 된다. 의뢰자는 그 요청이 어떻게 수행될것인가를 모르며 또 알도록 허가되지 않는다. 일단 요청이 수행되면 조종은 의뢰자에게 넘겨 진다. 이 시점에서 의뢰자가 알게 되는것은 요청이 수행되였다는것이며 이것이 어떻게 달성되였는가에 대한 표상은 전혀 없다.

이러한 기준들을 이제 승강기문제의 실례연구에 적용한다. 세부클라스도(그림 13-14)는 작용(방법)들을 그림 12-9의 클라스도에 추가하여 얻어 진다(Java실현의 경우에

두개의 추가적인 클라스들이 필요하다. **Elevator Application**은 C++ 기본함수에 대응하며 **Elevator Utilities**는 C++클라스들의 밖에서 선언된 C++ 함수들에 대응하는 Java루틴을 포함한다.).

승강기조종기에 대한 CRC카드의 두번째 반복을 고찰하자(그림 12-8). 책임들은 두개의 그룹으로 된다. 책임 8. Start timer, 10. Check requests, 1. Update requests는 책임구동설계에 기초하여 승강기조종기에 할당되였다. 즉 이 과제들은 승강기조종기 그자체에 의하여 수행된다.

한편 나머지 8개의 책임(사건 1부터 7까지와 사건 9)들은 《통보를 다른 클라스에 보내여 그에게 무엇을 하는가를 알려 주라.》는 형식을 취한다. 이것은 클라스에 련관된 작용을 할당할 때 리용되여야 하는 원리는 또다시 책임구동설계로 되여야 한다는것을 암시하고 있다. 이밖에 안전성과 관련하여 정보은폐의 원리는 8개의 경우에 모두 동등하게 응용될수 있다.

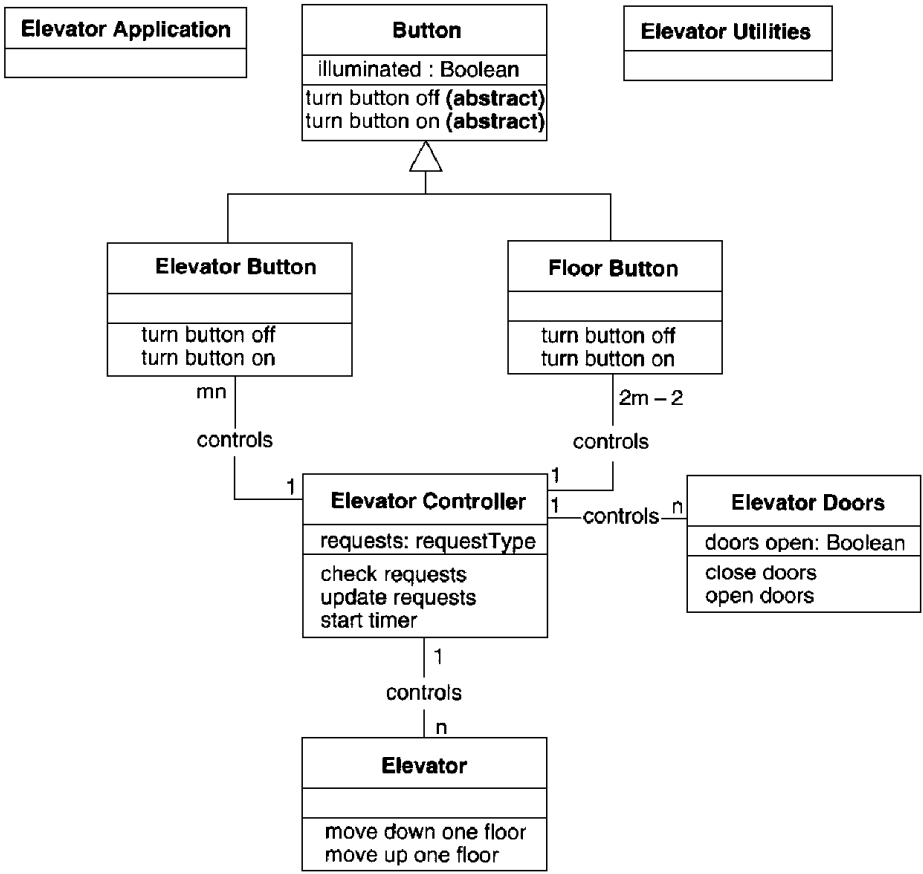



그림 13-14. 승강기문제의 세부클라스도

이상의 두가지 리유로 하여 방법 close doors와 open doors는 클라스 **Elevator Controller**

에 할당되였다. 즉 **Elevator Doors**의 한 의뢰자(이 경우에 클라스 **Elevator Cotroller**의 하나의 실례)는 클라스 **Elevator Doors**의 한 객체에 통보를 보내여 승강기의 문을 닫든가 연다. 이 요청은 련관된 방법에 의해서만 수행된다. 이 두 방법들의 매개 측면은 클라스 **Elevator Doors**내에 교갑화된다. 더우기 정보은페는 실지로 독립적인 하나의 **Elevator Doors**클라스를 생성시키는데 이것은 상세설계와 실현을 독립적으로 거쳐서 후에 다른 제품에서 재리용될수 있다.

완전히 동일한 두개의 설계원리들이 방법 move down one floor와 move up one floor에 적용되였는데 이 방법들은 클라스 **Elevator**에 할당되였다. 승강기를 멈추도록 명백한 지령을 줄 필요가 없다. 만일 클라스 **Elevator**의 두 방법들이 호출되지 않으면 승강기는 움직일수 없다. 이 두 방법들중의 하나를 호출하지 않고 승강기의 상태를 변화시킬수 있는 다른 방법은 없다.

마지막으로 방법 turn button off와 turn button on은 두 클라스 **Elevator Button**과 **Floor Button**에 모두 할당되였다. 그 리유는 클라스 **Elevator** Doors와 클라스 **Elevator**에 방법들이 할당된것과 마찬가지이다. 첫째로, 책임구동설계원리는 단추들이 켜지든가 꺼지든가에 대하여 완전히 조종할수 있을것을 요구한다. 둘째로, 정보은페원리는 어떤 단추의 내부상태가 은페될것을 요구한다. 승강기단추를 켜든가 끄는 방법들은 클라스 **Elevator Button**에 국한되여야 하며 클라스 **Elevator Button**에 대해서도 이와 류사하다. 다형성과 동적결합을 리용하기 위하여 방법 turn button on과 turn button off는 7.8에서 설명한 리유로 하여 기초클라스 **Button**안에서 추상적(가상적)방법으로 선언된다. 실행시에 방법 tun button on의 정확한 판본이 호출될것이다.

3단계. 객체들의 의뢰자들에 의하여 제품을 설계한다 객체 O에 하나의 통보를 보내는 하나의 객체 C는 O의 하나의 의뢰자로 된다. 호상작용도들은 매 객체의 의뢰자들을 보여 주는 선도를 작성하는데 리용된다. 의뢰자 C로부터 O에로 보내질 하나의 통보는 C로부터 O로 향하는 하나의 화살표로써 표현된다.

그림 13-15(승강기문제의 C++실현을 위한것)와 그림 13-16(Java실현을 위한것)은 각이한 클라스들에 대한 협동도들로부터 직접 구성된다. 실례로 그림 13-11의 대본(그림 13-14)에 대한 협동도에서 클라스 **Elevator Controller**의 한 객체는 클라스 **Elevator Controller**, **Elevator Button**, **Elevator Doors** 및 **Elevator**의 객체들에 통보를 보낸다. 또한 클라스 **Elevator Button**과 **Elevator Button**의 객체들은 클라스 **Elevator Controller**의 하나의 객체에 통보들을 보낸다.

어떤 다른 객체의 의뢰자로 되지 않는 임의의 객체는 일반적으로 주프로그람에 의하여 시작될 필요가 있다. 승강기문제의 C++실현에서 함수 main은 객체 **elevator controller**를 호출하여야 한다. Java제품은 객체 **elevator application**을 호출함으로써 동작을 시작한다. 이러한 형식의 구조는 객체지향파라다임이 리용될 때 자주 쓰인다. 즉 간단한 주프로그람이 모든것들을 출발시키고 그다음 객체 그자체들이 넘겨 받는 형식의 구조이다.

이제 전반적인 구성방식설계가 명확해 졌다고 하면 매 객체들이 요구하는 방법들을 모두 가지고 있는가를 검열하는것이 필요하다. 즉 객체에 보내는 매 가능한 통보들에 객체가 응답할수 있도록 매개의 객체들에 요구되는 모든 방법들이 할당되였다는것을 검증하는것이 필요하다. 이제 **elevator controller**가 main(또는 **elevator application**)이 호출할수

있는 어떤 추가적방법을 요구한다는것은 명백하다. 그러므로 그림 13-14는 수정되며 방법 elevat o r event loop가 클라스 **Elevator Controller**에 추가된다. 그러면 이 방법은 main(또는 **elevator application**)에 의하여 호출될수 있다.

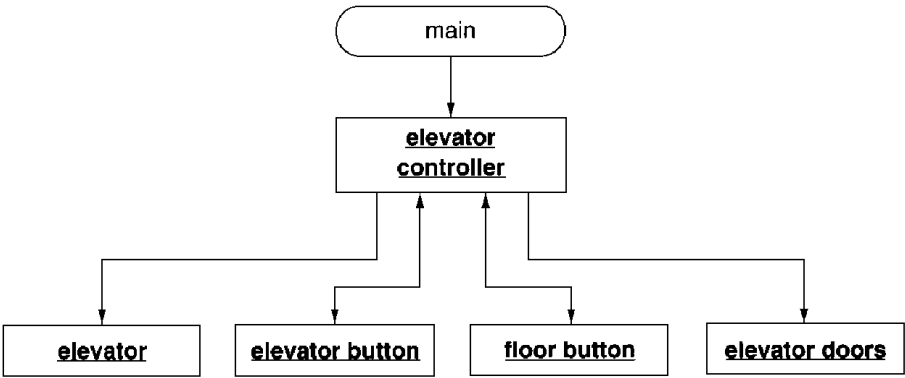



그림 13-15. (C++ 실현을 위한)의뢰자-객체관계

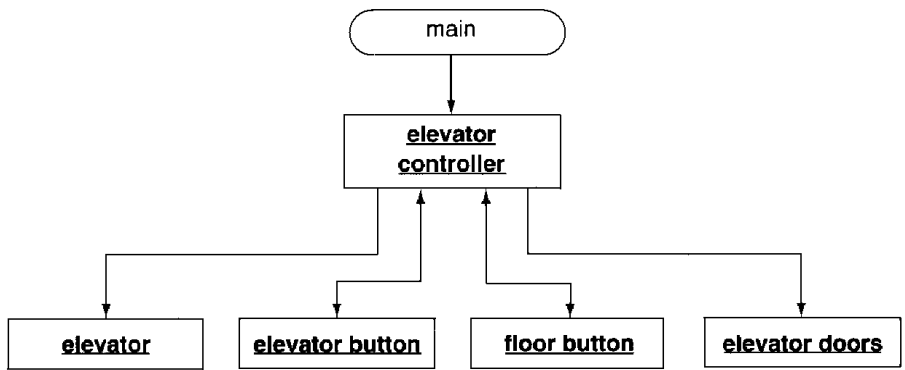



그림 13-16. (Java 실현을 위한)의뢰자-객체관계

4단계. 상세설계를 진행한다 이제 상세설계가 모든 클라스들에 대하여 전개된다. 제5장에서 서술한 계단식세련과 같은 임의의 적당한 기법들이 리용될수 있다. 방법 elevator event loop의 상세설계를 그림 13-17에 보여 주었다. 여기서는 PDL(의사코드)이 리용되였지만 표현(그림 13-6에서와 같은)방법도 마찬가지로 효과적일수 있다.

그림 13-17은 그림 12-6의 상태로부터 구성된다. 실례로 표식기호 [button pushed, button unlit]는 그림 13-17의 서두에서 두개의 함유 **if**명령문으로써 실현된다. 그다음 상태 **Process Requests**의 두개의 작용들이 제시된다. **else-if** 조건문은 **Elevator Event loop**의 다음표식기호[elevator moving in direction d, floor f is next]에 대응한다. 상세설계의 나머지부분들은 모두 단순하다.

```
void elevator event loop(void)
{
    while (TRUE)
    {
      if (button  has been pressed)
        if (button  is not on)
        {
              update requests;
              button::turn button on;
         }
      else if (elevator is moving  up)
      {
        if (there is no request to stop at floor f)
            elevator:: move one floor up;
        else
        {
            stop  elevator by not sending a message to move;
            elevator doors:: open doors;
            start timer;
            if (elevator button is on)
              elevator button:: turn button off;
            update requests;
         }
      }
      else if ( elevator is moving down)
        [ similar to up case ]
      else if ( elevator  is stopped and request is pending)
      {
          elevator doors:: close doors;
          if (floor button  is on)
            floor button:: turn button off;
           determine direction of next request;
          elevator:: move one floor up/down;
       }
       else if ( elevator is at rest and not (request is pending))
          elevator doors:: close doors;
       else
          there are no requests,  elevator is stopped with  elevator doors closed, so do nothing;
  }
}
```

그림 13-17. 방법 elevator event loop의 상세설계

객체지향설계의 단계들을 아래의 란에 종합하였다.

객체지향설계를 진행하는 방법

- 매 대본에 대하여 호상작용도들을 구성한다.
- 세부클라스도를 작성한다.
- 객체들의 의뢰자들에 의하여 제품을 설계한다.
- 상세설계를 진행한다.

１３．８．상세설계를 위한 형식적기법

상세설계를 위한 한가지 기법은 이미 제시되였다. 5.1에서 계단식세련방법이 서술되였다. 계단식세련방법은 그다음 흐름도표를 리용하여 상세설계에 적용되였다. 계단식세련방법이외에 형식적기법들이 상세설계의 개선에 리용될수 있다. 제6장은 완전한 제품을 실현하고 그다음 그것이 정확하다는것을 증명하는것은 비생산적일수 있다는것을 제기하고 있다. 그러나 증명과 상세설계를 병행으로 진행하는것과 코드를 주의 깊게 시험하는것은 상당히 차이나는 문제이다. 상세설계단계에서 형식적기법들은 다음 세가지 방법으로 도움을 줄수 있다.

1. 정확성증명에서의 최신기법은 비록 그 기법이 일반적으로 제품전체에 적용될수 없다고 하더라도 제품의 모듈크기의 부분들에는 적용될수 있다는것이다.
2. 증명을 상세설계와 함께 전개하는것은 정확성증명을 리용하지 않을 때보다 적은 오유를 가진 제품을 개발하는데로 이어 져야 한다.
3. 많은 같은 프로그람작성자가 상세설계와 실행을 모두 책임진다면 흔히 있는 경우와 마찬가지로 그 프로그람작성자는 상세설계가 정확하다고 확신할것이다. 제품에 대한 이러한 결정적인 태도는 보다 적은 오유를 가진 코드작성에로 이어 져야 한다.

１３．９．실시간설계기법

6.4.4에서 설명한바와 같이 실시간쏘프트웨어는 장치적인 시간제약 즉 어떤 제약이 만족되지 않으면 정보가 잃어 진다는 성질과 같은 시간제약에 의해 특징 지어 진다. 특히 매개 입력은 다음 입력이 당도하기전에 수행되여야 한다. 이와 같은 체계의 한가지 실례는 콤퓨터조종핵반응기이다. 핵의 온도와 반응기물준위와 같은 입력들은 콤퓨터에 련속적으로 보내 지는데 콤퓨터는 매 입력값을 읽고 다음입력이 오기전에 필요한 처리를 진행한다. 또 한가지 실례는 콤퓨터조종집중치료기이다. 여기에는 두가지 류형의 환자자

료가 존재한다. 즉 매 환자의 심장박동, 체온, 혈압과 같은 일상정보와 환자의 상태가 경각에 이르렀다는것을 체계가 알았을 때의 비상정보가 있다. 이와 같은 비상사태들이 발생하였을 때 쏘프트웨어는 한명 또는 그이상의 환자들에게서 오는 일상입력들과 비상사태관련입력들을 모두 처리하여야 한다.

많은 실시간체계들의 한가지 특성은 그것들이 분산된 하드웨어상에서 실현된다는것이다. 실례로 한대의 전투기를 조종하는 쏘프트웨어는 5대의 콤퓨터상에서 실현될수 있다. 한대는 비행을 조종하고, 한대는 무기계통을 조종하며, 세번째는 전자대응수단을 위한것이며, 네번째 콤퓨터는 보조날개와 기관들과 같은 비행기장치들을 조종하며, 다섯번째 콤퓨터는 전투에서 전술방안을 제안한다. 하드웨어를 전적으로 믿을수는 없기때문에 오동작하는 장치를 자동적으로 교체하는 반결합콤퓨터들을 추가할수 있다. 이와 같은 체계의 설계는 중요한 통신적인 의미를 가질뿐만아니라 앞에서 설명한 류형의 시간에 관한 문제들이 체계의 분산성의 결과로써 발생하게 된다. 실례로 전투조건하에서 전술작성콤퓨터는 조종기에 상승할것을 제기하며 반면에 무기계통콤퓨터는 어떤 특정한 무기가 최량조건하에서 발사될수 있도록 조종기에 급강하하도록 권고한다. 그러나 조종사는 조종간을 오른쪽으로 움직이기로 결심하며 따라서 비행기가 지적된 방향으로 경사 지어 비행하도록 비행장치콤퓨터에 신호가 전송되여 필요한 조절이 진행된다. 이 모든 정보들은 비행기의 실제적운동이 제안된 전술방안보다 모든 방법에서 우위를 차지하게 하는 방법으로써 처리되여야 한다. 더우기 비행기의 실제적운동은 새로운 제안들이 제안된 조건이 아니라 실제적조건들에 비추어 공식화할수 있도록 전술 및 무기계통콤퓨터들에 중계되여야 한다.

실시간체계가 분산된 하드웨어상에서 실현되게 된다고 하자. 두개의 작용들이 하나의 자료항목을 배타적으로 리용하며 또 하나의 작용이 다른 자료항목들을 배타적으로 리용하게 되는 경우에 교착과 같은 정황이 발생한다. 물론 교착상태는 분산된 하드웨어상에서 실현되는 실시간체계에서만 발생하는것은 아니다. 그러나 교착은 입력의 순서나 박자를 조종하지 않는 실시간체계들에서 특별히 시끄러우며 정황은 하드웨어의 분산성으로하여 복잡해 진다. 교착상태외에도 경쟁조건들을 비롯한 다른 동기화문제들도 발생할수 있다. 자세한 내용을 알기 위하여서는 문헌 [Silberschatz and Galvin, 1998]이든가 다른 조작체계관련교재들을 참고할수 있다.

이러한 실례들로부터 실시간체계의 실례와 관련한 중요한 난점은 시간제약들이 그 설계에 의하여 만족된다는것을 확인하는것이라는것이 명백해 진다. 즉 설계기법들은 그것이 실현될 때 설계가 요구되는 속도로 들어 오는 자료를 읽고 처리할수 있다는것을 검사하기 위한 기구를 제공하여 주어야 한다. 더우기 설계에서 동기화문제들이 정확하게 제기되였다는것도 보여 줄수 있어야 한다.

콤퓨터시대가 시작된 이래 하드웨어기술에서의 진보는 거의 모든 측면에서 쏘프트웨어기술에서의 진보를 릉가하였다. 그러므로 앞에서 설명한 실시간체계들의 모든 측면을 조종할수 있는 하드웨어는 존재하여도 쏘프트웨어설계기술은 상당히 뒤떨어 져 있다. 실례로 제11장과 제12장에서 서술한 대다수의 명세서작성기법들은 실시간체계들을 명시하는데 리용될수 있다. 그러나 쏘프트웨어설계는 아직 이러한 세련된 준위에 도달하지 못하였다. 큰 진보는 이룩되였지만 최신성에 있어서는 명세작성기술에서 도달된 성과와는

대비할수 없다. 실시간체계를 위한 거의 모든 설계방법들을 선택할수 있는 기법이 전혀 없기때문에 많은 실시간체계들을 경험적으로 리용하고 있다. 그러나 앞에서 서술한것과 같은 실시간체계들을 설계하며 체계가 실현되기전에 매 실시간제약들이 만족되며 동기화 문제들이 발생하지 않을것이라는것을 확정할수 있게 되려면 아직 멀었다.

대다수 실시간설계기법들은 비실시간기법들의 실시간령역에로의 확장이다. 실례로 실시간체계에 대한 구조화된 개발(SDRTS)[Ward and Mellor, 1985]는 본질에 있어서 구조화체계분석(11.3), 자료흐름분석(13.3), 트랜잭션분석(13.4)의 실시간쏘프트웨어에로의 확장이다. 이 개발기법은 실시간설계를 위한 한가지 요소를 포함하고 있다. 레비(Levi) 와 아그라왈라(Agrawala)는 정보은페(7.6)의 개념에 기초하여 실시간방법들을 개발하였다[Levi and Agrawala, 1990]. 앞에서 서술한바와 같이 유감스럽게도 실시간체계의 최신기법은 우리가 바라는것만큼 진보하지 못하였다.

그러나 문헌 [Stankovic, 1995]에서 서술하고 있는것처럼 이 정황을 개선하기 위하여 현재 상당한 노력을 기울여 연구를 진행하고 있다.

13. 10. 설계단계에서의 시험

설계단계에서 시험의 목적은 설계 그자체의 정확성을 확인하는것과 함께 명세서가 설계에 정확하고 완전하게 병합되였다는것을 검증하는것이다. 실례로 설계에 론리적오유들이 없어야 하며 모든 대면부들이 정확히 정의되여야 한다. 설계에서의 모든 오유들을 코드작성이 시작되기전에 발견하는것이 중요하다. 그렇지 않으면 그 오유를 수정하는데 드는 비용이 상당히 높아 지게 될것이다. 설계오유들은 설계관통심사회의뿐아니라 설계검토에 의하여 발견될수 있다. 설계검토는 이 절의 나머지부분에서 론의되는데 설계관통심사회의에도 마찬가지로 주목을 돌려야 한다.

제품이 트랜잭션을 위한것일 때(13.4) 설계검토는 트랜잭션을 반영하여야 한다 [Beizer, 1990]. 모든 가능한 류형의 트랜잭션을 포함하는 검토계획이 작성되여야 한다. 심사자는 설계안의 매 트랜잭션들을 명세서와 관련시켜 그 트랜잭션들이 명세서로부터 어떻게 생성되는가를 보여 주어야 한다. 실례로 만일 응용제품이 자동출납기라면 하나의 트랜잭션은 신용카드에 예금을 한다든가 또는 신용카드에서 자금을 대출한다든가와 같은 손님이 진행할수 있는 매 조작에 대응한다. 다른 실례들에서 명세서와 트랜잭션사이의 대응은 필수적으로 1대 1 대응되지는 않는다. 실례로 교통신호등조종체계에서 만일 한대의 자동차가 수감구역으로 들어 섬으로써 조종체계가 어떤 특정한 등을 15s내에 붉은색으로부터 푸른색으로 바꾸도록 결정하게 한다면 수감구역으로부터 오는 이 두 신호들은 무시될수 있다. 거꾸로 교통흐름량을 늘이기 위하여 수감구역에서 오는 하나의 신호가 모든 교통신호들을 붉은색에서 푸른색으로 바꾸도록 할수도 있다.

검토를 트랜잭션구동검토에 제한시키면 설계자들이 명세서에서 요구되는 트랜잭션실례들을 못 보고 지나친 경우를 발견하지 못하게 될것이다. 극단한 실례로서 교통신호등조종기에 대한 명세서들은 저녁 11h부터 아침 6h사이에는 모든 신호등들이 한 방향에서는 노란색으로 다른 방향에서는 붉은색으로 켜지게 된다는것을 명기할수 있다. 만일 설

계자들이 이 명기내용을 못 보고 지나치게 되면 저녁 11h와 아침 6h에서의 박자발생트랜잭션은 설계에 포함되지 않게 될것이다. 그리고 이 트랜잭션들을 놓치게 되면 그것들은 트랜잭션에 기초한 설계검토에서 시험되지 않게 될것이다. 그러므로 트랜잭션구동식설계검토계획을 작성하는것은 적당치 않다. 명세서안의 설명문들이 하나도 스쳐 지났거나 잘못 해석되지 않았다는것을 확인하는데서는 명세서구동검토가 역시 기본으로 된다.

13. 11. 설계단계를 위한 CASE도구

앞절에서 설명한바와 같이 설계단계에서 한가지 중대한 국면은 설계문서가 명세서의 모든 측면들을 정확하게 병합하고 있다는것을 시험하는것이다. 그러므로 명세서와 설계문서에 모두 리용될수 있는 CASE도구 즉 앞단(*front-end*) 또는 상위(*upper*)CASE도구들이 요구된다(이 도구들은 실현, 통합, 유지정비를 지원하는 뒤단(*back-end*) 또는 하위(*lower*) CASE도구들이다.).

많은 상위CASE도구들이 시장에서 판매되고 있다. 보다 대중적인것들로서는 Analyst/Designer, Software through Pictures, System Architect들이다. 상위CASE도구들은 일반적으로 자료사전을 중심에 두고 구축된다. 이 CASE도구들은 사전안의 기록의 매 마당이 설계문서의 어디에서 언급되고 있는가 또는 설계문서의 매 항목이 자료흐름도에 반영된다는것을 검사할수 있다.

더우기 대다수 상위CASE도구들은 일관성검사기를 병합하고 있는데 이것은 자료사전을 리용하여 설계안의 매 항목이 명세서에 선언되였으며 거꾸로 명세서안의 매 항목이 설계에 나타난다는것을 결정한다. 또한 대다수 상위CASE도구들은 화면구성 및 보고서생성프로그람들을 병합하고 있다. 즉 의뢰자는 어느 항목들이 어떤 보고서에 또는 어떤 입력화면에 나타나게 되며 매 항목들이 어디서 어떻게 나타나게 되는가를 명시할수 있다. 매개의 항목을 고려하여 완전한 세부들이 자료사전안에 있기때문에 CASE도구들은 보고서를 인쇄하거나 의뢰자의 요구에 따라 입력화면을 현시하기 위한 코드를 쉽게 생성할수 있다. 일부 상위CASE제품들은 평가 및 계획작성을 위한 관리도구들을 병합할수 있다.

객체지향설계에 관하여 Together, Rose, Software through Pictures들은 완전한 객체지향생명주기의 범위내에서 이 단계를 지원하여 준다.

13. 12. 설계단계를 위한 척도

설계의 측면들을 서술하기 위하여 각이한 척도들을 리용할수 있다. 실례로 모듈의 개수는 목적하는 제품의 크기에 대한 하나의 자연스러운 척도로 된다. 모듈의 응집도와 결합도는 오유통계량과 마찬가지로 설계의 질에 관한 척도로 된다. 다른 모든 류형의 검사와 마찬가지로 설계검토기간에 발견된 설계오유의 개수와 류형에 대한 기록을 보관하는것은 매우 중요하다. 이 정보는 제품의 코드검토기간과 련이은 제품들의 설계검토에서 리용된다.

상세설계의 주기적복잡도 M은 2분결정(술어)수에 1을 더한 수이며 [McCabe, 1976]

또는 등가적으로 모듈에서의 가지의 개수이다. 주기적복잡도가 설계의 질에 대한 하나의 척도로 된다는것이 제시되였다. *M*의 값이 작을수록 더 좋다. 이 척도의 우점은 계산하기 쉽다는것이다. 그러나 이 척도는 하나의 고유한 문제점을 가지고 있다. 주기적복잡도는 순전히 조종복잡도에 대한 측정값이며 자료의 복잡도를 무시하고 있다. 즉 *M*은 어떤 표안의 값과 같이 자료에 의하여 구동되는 모듈의 복잡도를 재지 못한다. 실례로 설계자가 C++서고함수 toascii를 모르고 사용자가 입력한 하나의 문자를 읽고 대응하는 ASCII코드(0과 127사이의 옹근수)를 귀환하는 모듈을 처음부터 설계한다고 하자. 이 모듈을 설계하기 위한 한가지 방법은 **switch**명령에 의하여 수행되는 128개의 가지를 리용하는것이다. 두번째 방법은 128개의 문자들을 ASCII코드순서로 포함하고 있는 하나의 배렬을 만들고 하나의 순환을 리용하여 사용자가 입력한 문자를 문자배렬의 매 요소와 비교하는것이다. 그러면 순환변수의 현재값은 대응하는 ASCII코드이다. 이 두가지 설계는 기능상 등가이지만 각각 128과 1의 주기적복잡도를 가진다.

구조화된 파라다임이 리용될 때 설계단계에 대한 어떤 련관된 부류의 척도는 구성방식설계를 하나의 방향그라프로 표현하는데 기초하고 있다. 이 방향그라프에서 모듈들은 마디점으로 표시되고 모듈(절차와 기능호출)들사이의 흐름들은 호로써 표시된다. 어떤 모듈의 입력수(*fan-in*)는 그 모듈에로의 흐름들의 개수에 그 모듈이 접근하는 대역자료구조의 개수를 더한것으로서 정의할수 있다. 이와 류사하게 출력수(*fan-out*)는 그 모듈로부터 나오는 흐름의 개수에 그 모듈에 의하여 갱신되는 대역자료구조의 개수를 더한것으로서 정의된다. 그러므로 모듈의 복잡도는 길이$\times$(입력수$\times$출력수)^{2}에 의해 주어진다. 여기서 길이(length)는 모듈의 크기에 대한 측정값이다.(9.2.1). 입력수와 출력수의 정의가 대역자료를 병합하고 있기때문에 이 척도는 자료의존요소를 가지고 있다. 그러나 경험은 이 척도가 주기적복잡도와 같은 보다 단순한 척도보다 더 좋은 복잡도측도로 되지 않는다는것을 보여주었다[kitchenham, Pickard, and Linkman, 1990, Shepperd, 1990].

설계척도에 대한 론의는 객체지향파라다임이 리용될 때 보다 복잡하게 된다. 실례로 많은 클라스들은 전형적으로 여러개의 작고 단순한 방법들을 포함하기때문에 한개 클라스의 복잡도는 일반적으로 낮다. 더우기 앞에서 지적한바와 같이 주기적복잡도는 자료의 복잡도를 무시하고 있다. 자료와 작용들은 객체지향파라다임내에서 동등한 상대자이기때문에 주기적복잡도는 어떤 객체의 복잡성에 영향을 줄수 있는 하나의 중요한 요소를 고찰하지 않고 있다. 그러므로 주기적복잡도로 병합하는 클라스들에 대한 척도는 일반적으로 적게 리용된다.

객체지향설계에 대한 여러가지 척도들이 제시되였다. 례하면 문헌[Chidamber and Kemerer, 1994]에 제시된 척도들이다. 이 척도들과 기타 척도들은 리론적 및 실험적립장에서 연구되였다[Binkley and Schach, 1996; 1997; 1998].

1 3. 1 3. 항공음식전문회사 실례연구: 객체지향설계

13.6에서 서술한바와 같이 객체지향설계는 네단계로 구성된다.

1단계: 매 대본에 대한 호상작용들을 작성한다 예약에 대한 확장대본(그림 12-12)의 순차도를 그림 13-18에 보여 주었다. 이 순차도는 대본안의 매 설명문을 취하여 련관된 클라스들의 실례들사이에 화살표를 그음으로써 작성되였다.

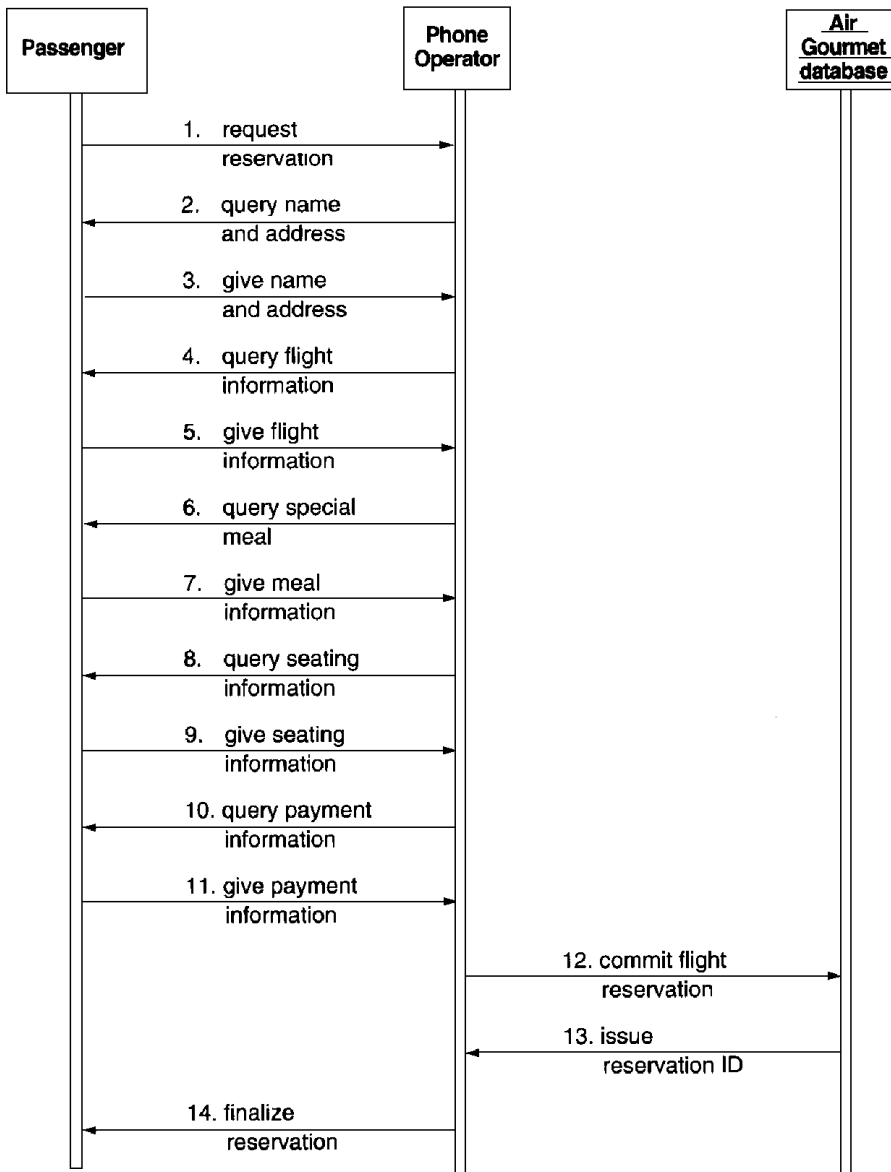



그림 13-18. 예약을 하는 대본을 위한 순차도(그림 12-12)

보다 세부적으로 말하면 먼저 그 대본의 작용자들 즉 **Passenger**, **Phone Operator**, **<u>Air Gourmet database</u>**가 결정되였다.

이 세개의 작용자들의 이름을 포함하고 있는 통들은 그림 13-18의 상단에 배치되며 수직2중선이 그려 진다. 이제 대본의 매 사건들이 조사되여 순차도에 들어 간다. 실례로 첫번째 사건에서 승객은 예약을 하기 위하여 교환수를 전화로 찾는다. 이 사건은 **Passenger** 수직2중선으로부터 **Phone Operator**수직2중선에로 향하는 표식 1. request reservation이 붙은 수평화살표로 모형화된다.

순차도의 나머지부분들도 다른 클라스들에 대한 순차도에서와 마찬가지로 간단하다. 우편엽서의 반환과 조사에 대한 대본(그림 12-14)의 협동도를 그림 13-19에 보여 주었다. 그림 13-18의 순차도와 마찬가지로 협동도를 구성할 때의 첫 단계는 확장대본(그림 12-14)을 조사하고 작용자들을 결정하는것이다. 이 경우에 작용자들은 **Air Gourmet Staff Member, Passenger, <u>Air Gourmet database</u>**로 찾아 냈다. 이 작용자들이 그림 13-19에 그려 진다. 다음으로 매 사건들이 조사되며 선도에 들어 간다. 사건 1에서 항공음식전문회사의 한 직원이 특별식사를 받은 매 승객들에게 우편엽서를 보낸다. 먼저 **Air Gourmet Staff Member**와 **Passenger**사이에 하나의 직선이 그려 진다. 그다음 회사직원이 우편엽서를 승객에게 보낸다는것을 지적하기 위하여 하나의 표식 붙은 화살표가 삽입된다.

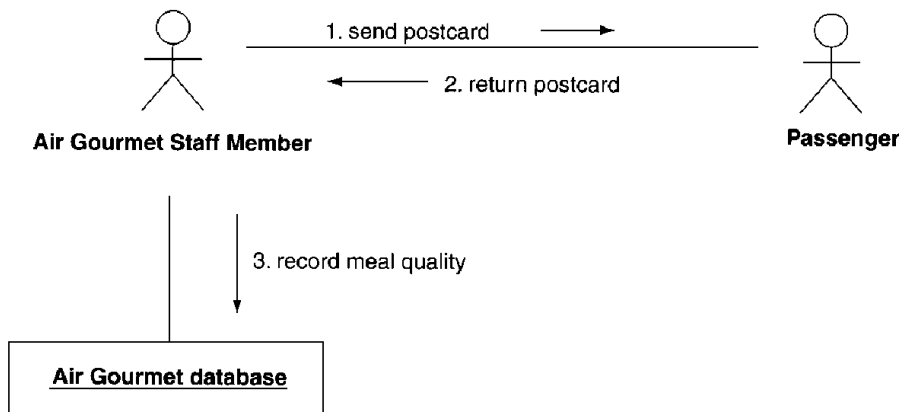



그림 13-19. 엽서의 반환과 조사에 대한 대본의 협동도(그림 12-14)

두번째 사건은 그 우편엽서를 받은 승객이 엽서에 답변을 채워 넣고 그것을 회사직원에게 돌려 보내는것이다. 이 사건은 표식 2. return postcard가 붙은 역방향화살표로 지적된다.

세번째 사건은 회사직원이 그 엽서를 받고 봉사 받은 특별식사에 대한 승객의 견해를 반영하기 위하여 련관된 비행기록을 갱신하는것이다. 항공음식전문회사직원을 표시하는 그림기호(*icon*)와 항공음식전문회사자료기지를 표시하는 통사이에 하나의 수직선이 그려 진다. 마지막으로 표식 3. record meal quality가 붙은 하나의 화살표가 추가된다. 나머지 호상작용도들도 이와 같은 방법으로 구성된다.

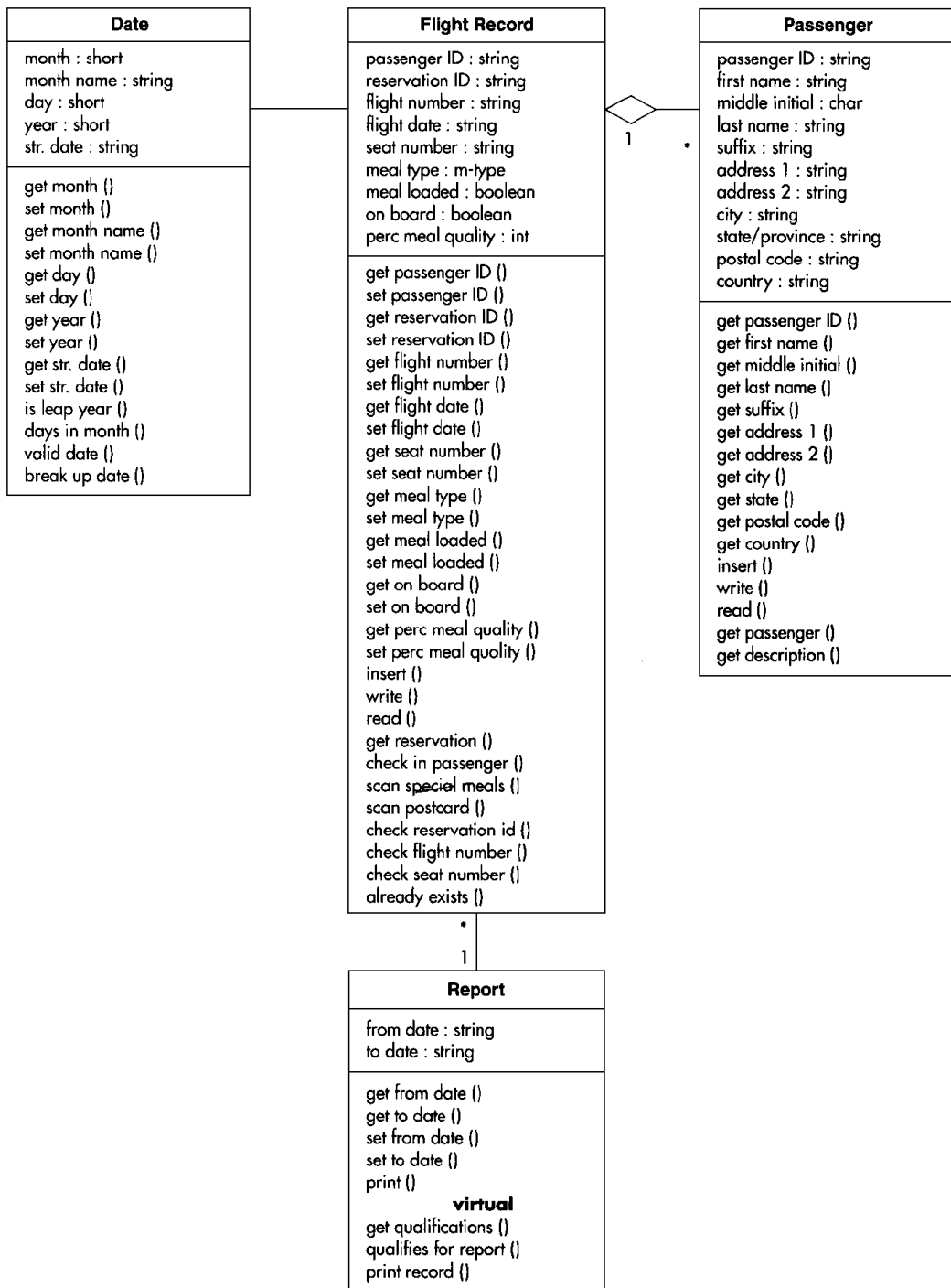



그림 13-20. 항공음식전문회사제품의 C++실현을 위한 세부클라스도

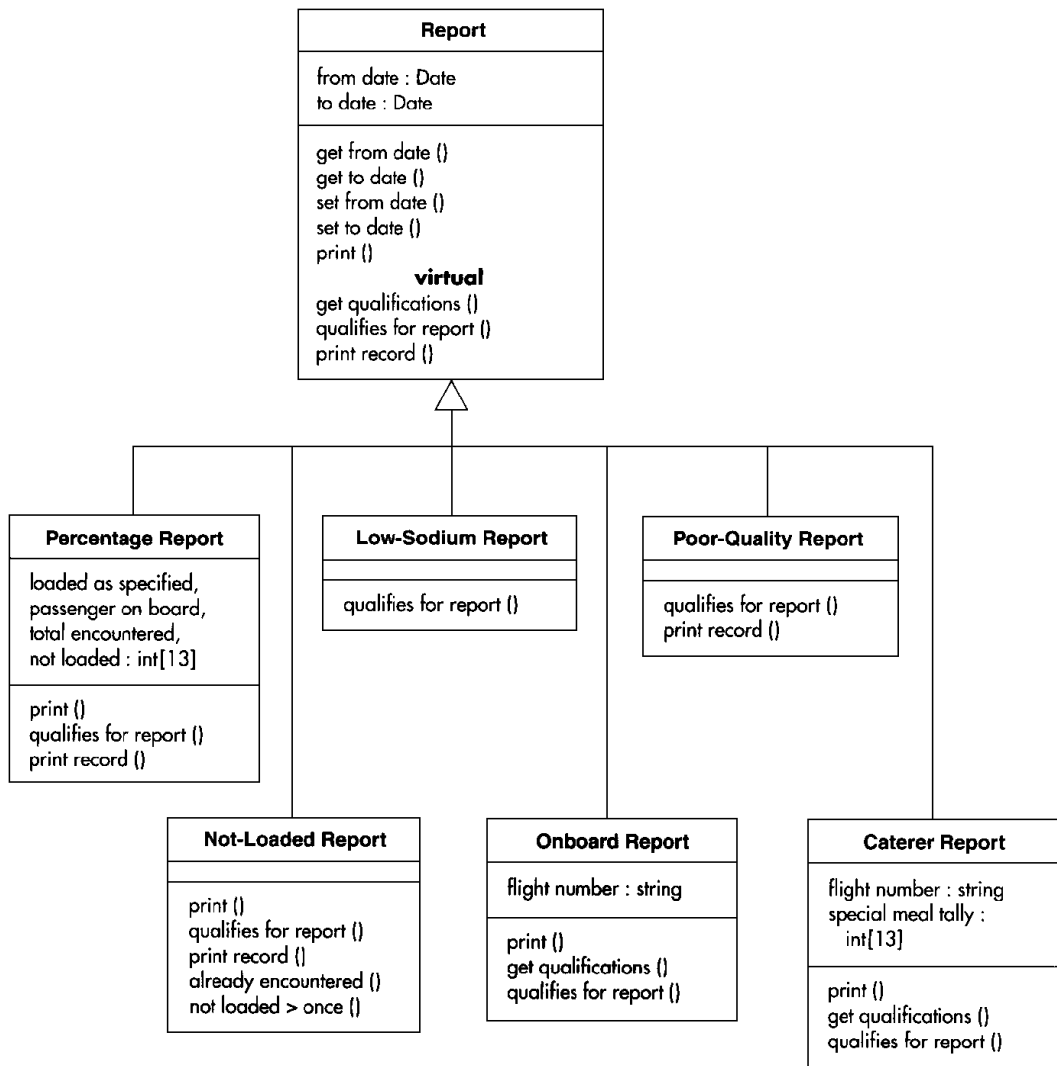



그림 13-20. 항공음식전문회사제품의 C++실현을 위한 세부클라스도(계속)

2 단계:세부클라스도를 구성한다 C++실현을 위한 세부클라스도를 그림 13-20 에 보여주며 Java 실현을 위한 세부클라스도는 그림 13-21 에 보여 준다. 두 그림사이의 한가지 차이는 Java 가 날자들을 처리하기 위한 내장된 클라스들 즉 **java.text.Dateformat** 와 **java.util.Calendar** 를 가지고 있으며 반면에 C++실현에서는 사용자정의클라스 **Date** 가 필요하다는것이다. 또 한가지 차이는 Java 는 순수한 객체지향언어이며(즉 Java 프로그람은 클라스도의 모임이다.) 반면에 C++는 함수들을 지원하고 있다는것이다. 이러한 차이와 관련하여 클라스 **Air Gourmet Application** 은 C++ 함수 main 에 대응하며 **Air Gourmet Utilities** 는 C++편의함수들에 대응한다.

Air Gourmet Application

Air Gourmet Utilities

get character ()
read string ()
clear screen ()
press enter ()
display main menu ()
display report menu ()

Flight Record

passenger ID : string
reservation ID : string
flight number : string
flight date : Date
seat number : string
meal type : m-type
meal loaded : boolean
on board : boolean
perc meal quality : int

get passenger ID ()
set passenger ID ()
get reservation ID ()
set reservation ID ()
get flight number ()
set flight number ()
get flight date ()
set flight date ()
get seat number ()
set seat number ()
get meal type ()
set meal type ()
get meal loaded ()
set meal loaded ()
get on board ()
set on board ()
get perc meal quality ()
set perc meal quality ()
insert ()
write ()
read ()
get reservation ()
check in passenger ()
scan special meals ()
scan postcard ()
check reservation id ()
check flight number ()
check seat number ()
already exists ()

1 *

Passenger

passenger ID : string
first name : string
middle initial : char
last name : string
suffix : string
address 1 : string
address 2 : string
city : string
state/province : string
postal code : string
country : string

get passenger ID ()
get first name ()
get middle initial ()
get last name ()
get suffix ()
get address 1 ()
get address 2 ()
get city ()
get state ()
get postal code ()
get country ()
insert ()
write ()
read ()
get passenger ()
get description ()

*
1

Report

from date : Date
to date : Date

get from date ()
get to date ()
set from date ()
set to date ()
print ()
abstract
get qualifications ()
qualifies for report ()
print record ()

그림 13-21. 항공음식전문회사제품의 Java실현을 위한 세부클라스도

그림 13-21. 항공음식전문회사제품의 Java실현을 위한 세부클라스도(계속)

제품에 대한 방법들은 각이한 호상작용선도들에 나타난다. 설계자의 과제는 매 방법들이 어느 클라스에 할당되여야 하는가를 결정하는것이다.

항공음식전문회사제품의 경우에 클라스도에로의 방법들의 할당은 단순하다. 실례로 방법 get passenger ID는 명백히 클라스 **Passenger**에 속한다.

3단계:객체들과 그의 의뢰자들에 의하여 제품을 설계한다 OOD의 세번째 단계는 13.7에서 보여 준것처럼 객체들과 그 의뢰자들에 의하여 제품을 설계하는것이다. 의뢰자-객체관계는 그림 13-22(C++실현)와 그림 13-23(Java실현)에 의뢰자-객체관계선도들은 매 CRC카드들을 조사하며 다른 클라스들의 어느 방법들이 그 클라스의 방법들에 의하여 호출되는가를 결정하는것으로써 구성된다. 일반적으로 만일 CRC카드들이 만들어 지지 않

았다면 클라스선도들이 이 목적에 리용될수 있다. 항공음식전문회사제품의 신속원형(부록 3과 4)은 차림표구동설계가 실현가능하다는것을 보여 주었다. 주차림표는 사용자가 여섯가지 보고서들가운데서 어느것을 인쇄하겠는가를 선택할수 있게 한다. 그림 13-22와 13-23에 이 내용이 반영되여 있다. 보여 준다. 이것들은 UML의 선도가 아니다. 경험에 의하면 이러한 종류의 선도들이 제품의 각이한 부분들이 어떻게 서로 조화되는가를 리해하는것을 도울수 있으며 특히 상세설계를 구성하기 쉽게 할수 있기때문에 이 선도들을 이 책에 포함시켰다.

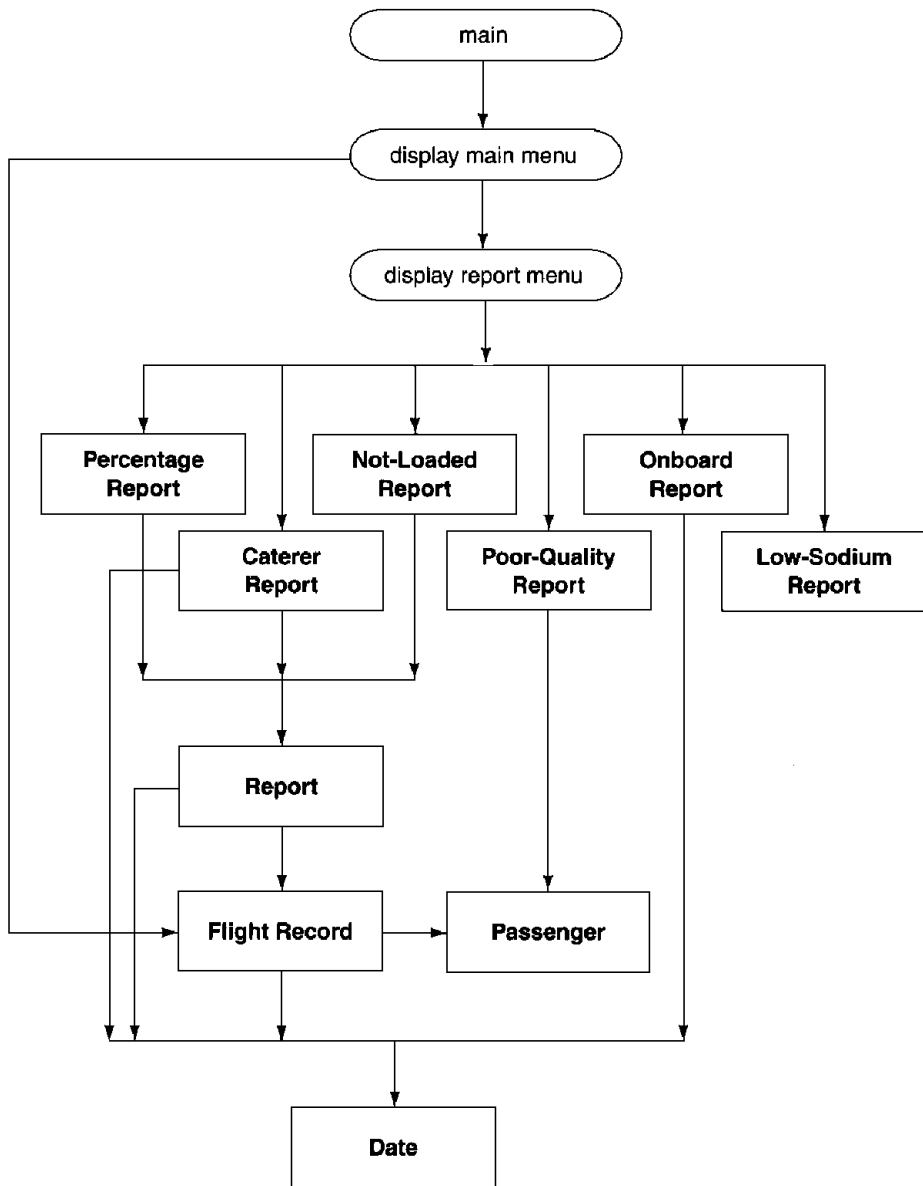



그림 13-22. 항공음식전문회사제품의 C++실현을 위한 의뢰자-객체관계

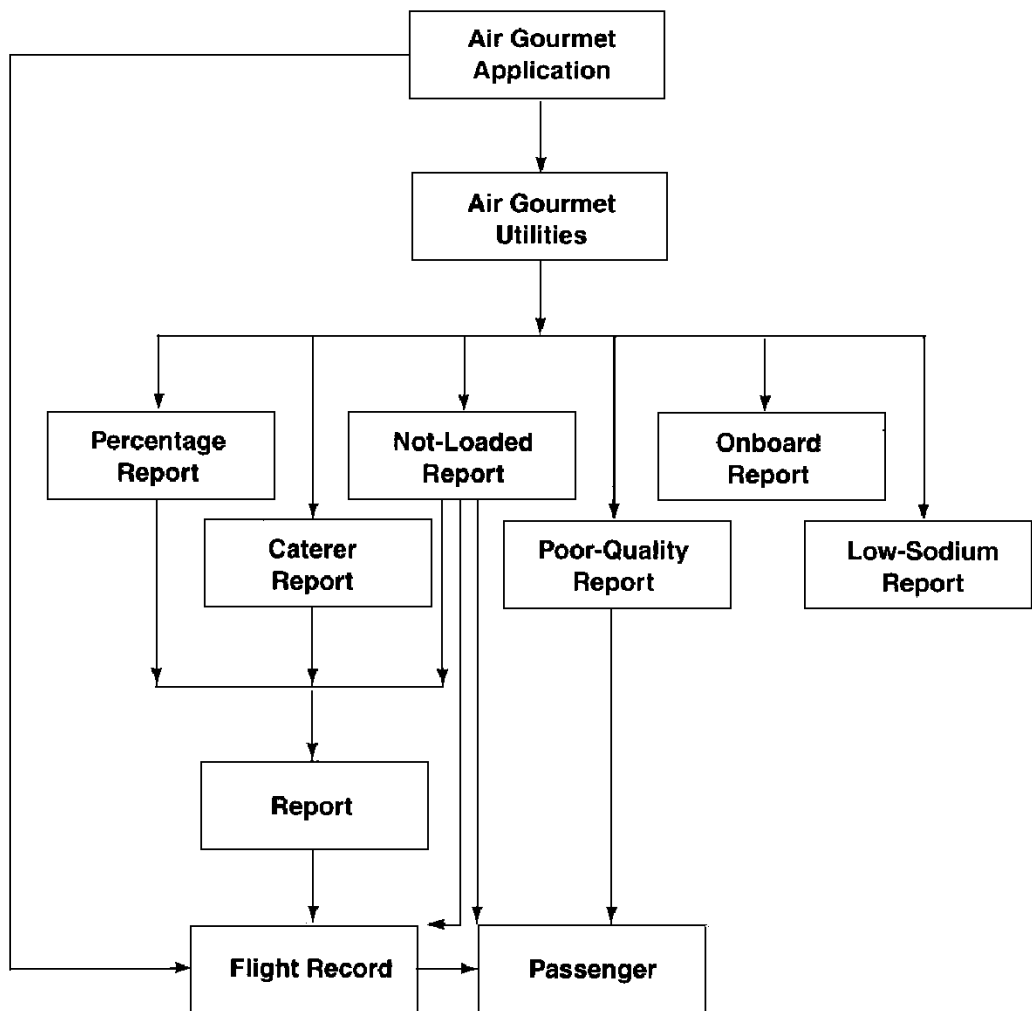



그림 13-23. 항공음식전문회사제품의 Java실현을 위한 의뢰자-객체관계

4단계: 상세설계를 진행한다 마지막으로 상세설계가 작성된다. 완전한 상세설계가 부록 8(C++실현)과 부록 9(Java실현)에 제시된다. 이 상세설계는 매 방법 또는 기능을 조사하며 그것이 무엇을 수행하는가를 결정하는것으로써 작성되였다. 상세설계에 대한 표식법은 약간 시끄러운것 같지만 그것의 형식성은 프로그람작성자들에게 도움이 된다. 이제는 설계가 외견상으로는 완전한것 같은데 이 설계의 모든 측면들은 재검사되여야 한다. 오유는 전혀 발견되지 않는다. 그러나 항공음식전문회사제품이 실현되고 종합될 때 아마도 이 설계가 근본적으로 다시 변경될 가능성이 있다.

１３．１４．설계단계에서의 난관

11.6과 12.9에서 지적한바와 같이 명세작성단계에서 너무 많은것을 하지 않는것이 중요하다. 즉 명세작성팀은 설계단계의 부분들을 때 이르게 시작하지 말아야 한다. 설계단계에서 설계팀은 다음의 두가지 방식으로 잘못해 나갈수 있다. 즉 너무 많이 하거나 너무 적게 하는것이다.

그림 13-7의 PDL이 아니라 C++ 또는 Java로 작성하려는 유혹은 아주 강하다. 즉 상세설계를 의사코드로 기술하는것 대신에 설계자가 모듈을 거의 모두 코드작성할수도 있다. 이것은 모듈의 륜곽을 서술하는것보다 그것을 작성하는데 더 많은 시간이 들게 하며 설계에서 오유가 발견되는 경우에 그것을 수정하는데 더 오랜 시간이 걸리게 한다(그림 1-5를 보시오.). 명세서작성팀과 마찬가지로 설계팀의 성원들은 자기들에게 필요한것보다 더 많은것을 하려는 충동을 억제하여야 한다.

동시에 설계팀은 너무 적게 설계하지 않는데도 주의를 돌려야 한다. 그림 13-6의 표형식의 상세설계를 고찰하자. 만일 설계팀이 바쁘다면 상세설계를 설명칸으로 수축하기로 결심할수도 있다. 지어는 프로그람작성자들은 자기들이 상세설계를 진행하기로 결심할수도 있다. 이러한 결심들은 모두 잘못되였다. 상세설계를 진행하는 선차적인 리유는 모든 대면부들이 정확하다는것을 확인하는것이다. 설명통 그자체가 이 목적에 부합되지 않는다. 상세설계가 전혀 도움이 안되는것은 아니다. 그러므로 설계단계에서의 한가지 난관은 설계자들이 정확한 작업량을 수행하도록 하는것이다.

이밖에 보다 중요한 한가지 난관이 있다. 론문 《은총탄은 없다.》[Brooks, 1986]에서 브룩스(Brooks)는 이른바 설계대가 즉 설계팀의 다른 성원들보다 훨씬 뛰여 난 설계가의 결여를 비난하고 있다. 브룩스의 견해에 의하면 쏘프트웨어프로젝트의 성공은 그 설계팀이 설계대가에 의하여 지도되는가에 결정적으로 의존한다. 훌륭한 설계는 다음의 사실을 가르쳐 줄수 있다. 즉 큰 설계는 설계대가에 의해서만 작성될수 있는데 설계대가들은 《매우 드물다.》.

그러므로 한가지 난관은 설계대가들을 양성하는것이다. 그들을 될수록 빨리 찾아 내서(가장 훌륭한 설계가들이 반드시 가장 경험있는것은 아니다.) 어떤 책임자에게 배치하여 설계대가들의 조수들과 마찬가지의 공식적인 교육을 주며 다른 설계가들과 호상작용할수 있도록 하여야 한다.

요　약

설계단계는 구성방식설계, 그다음 상세설계로 이루어 진다(13.1). 세가지 기본설계방법 즉 작용지향설계(13.2), 자료지향설계(13.5), 객체지향설계(13.6)가 있다. 작용지향설계의 두가지 실례들인 자료흐름분석(13.3)과 트랜잭션분석(13.4)이 서술된다. 객체지향설계를 13.7의 승강기문제에 적용한다. 상세설계를 위한 기법들이 13.8에 제시된다. 실시간체계는 13.9에서 설명된다. 13.10에서 설계의 시험을 론의한다. 설계단계에서의 CASE도구들과 척도들은 13.11과 13.12에서 각각 론의한다. 이 장은 항공음식전문회사 실례연구(13.13)와 설계단계에서의 난관에 대한 론의(13.14)로 결속된다.

보 충

자료흐름분석과 트랜잭션분석은 문헌 [Gane and Sarsen, 1979, and Yourdon Constantine, 1979]에 서술되여 있다. 잭슨(Jackson)의 기법은 문헌 [Jackson, 1975]에서 서술하고 있다. 워니오(Warnier)의 연구에 흥미를 가지는 독자들에게는 문헌 [Warnier, 1976] 이 기본문헌으로 되고 있다. 오르(Orr)의 연구방법에 대하여서는 문헌 [Orr, 1981], 객체지향설계와 관련한 정보는 문헌 [Wirfs-Brock, Wilkerson and Wiener, 1990; Coad and Yourdon, 1991b; Shlaer and Mellor, 1992; and Jacobson, Booch and Rumbaugh, 1999]에서 찾아 볼수 있다. 객체지향설계와 관련한 여러가지 기법들에 대한 비교는 문헌 [Monarchi and Puhr, 1992, and Walker, 1992]에서 서술하고 있다. 객체지향과 구조화설계기법사이의 비교는 문헌 [Fichman and Kemerer, 1992]에서 보여 주었다.

형식적인 설계기법들은 문헌 [Hoare, 1987]에서 서술하였다.

설계과정의 검토와 관련한 초기론문은 [Fagan, 1976]이다. 검토기법과 관련하여 그 후의 발전적인 결과는 문헌 [Fagan, 1980]에서 서술하여 주었다. 사용자대면부 설계를 시험하기 위한 관통심사회의의 리용에 대하여서는 문헌 [Bias, 1991]에서 서술하였다. 실시간설계와 관련한 특정한 기법은 문헌 [Ward and Mellor, 1985; Levi and Agrawala, 1990; and Cooling, 1997]에서 찾아 볼수 있다. 네개의 실시간설계기법의 비교에 대하여서는 *IEEE Computer* 1995년 6월호와 함께 문헌 [Kelly and Sherif, 1992]에서 찾아 볼수 있다. 분산체계의 설계에 대하여서는 문헌 [Kramer, 1994]에서 서술하여 주었다. *IEEE Software* 1992년 9월/10월호에는 구성방식설계와 관련한 많은 기사들이 들어 있다.

설계단계를 위한 척도에 대하여서는 문헌 [Henry Kafura, 1981; Brandl, 1990; Henry and Selig, 1990; and Zage and Zage, 1993]에서 서술하고 있다. 객체지향설계의 척도에 대하여서는 문헌 [Chidamber and Kemerer, 1994, and Binkley and Schach, 1996]에서 론의하였다.

쏘프트웨어명세작성 및 설계에 관한 국제토론회 회보에서는 설계기법에 대한 가치있는 정보들을 제공하고 있다.

문 제

13.1. 문제 11.6에 대하여 먼저 DFD를 작성하면서 자료흐름분석을 리용하여 은행계산서의 정확성여부를 결정하는 제품을 설계하시오.

13.2. 프랜잭션분석을 리용하여 ATM(문제 8.9)을 조종하기 위한 쏘프트웨어를 설계하시오. 이 단계에서 오유조종능력은 생략하시오.

13.3. 이제 문제 13.2 에 대하여 작성하는 설계를 조사하고 오유조종을 수행하기 위한 모듈들을 추가하시오. 결과설계를 주의 깊게 조사하고 모듈들의 응집도와 결합도를 결정하

시오. 그림 13−10에 묘사된것과 같은 정황에 주의하시오.

13.4. 상세설계를 묘사하기 위한 두개의 서로 다른 기법들이 13.3.1(그림 13−6과 13−7)에 제시된다. 두 기법을 비교대조하시오.

13.5. 자동서고순환체계(문제 11.8)에 대한 자료흐름도를 먼저 작성하면서 자료흐름분석을 리용하여 순환체계를 설계하시오.

13.6. 트랜잭션분석을 리용하여 문제 13.5를 반복하시오. 두 기법가운데서 어느것이 보다 적당하다고 생각하는가?

13.7. 자동서고순환체계(문제 12.2)에 대한 객체지향분석으로부터 시작하여 객체지향설계를 리용하여 서고체계를 설계하시오.

13.8. 객체지향설계를 리용하여 ATM쏘프트웨어(문제 8.9)를 설명하시오.

13.9. (과정안상 목표) 문제 11.5 또는 12.9에 대한 명세서를 먼저 작성하고 브로드랜즈지역 아동병원제품(부록 1)을 설계하시오. 지도교원이 규정한 설계기법을 리용하시오.

13.10. (실례연구)자료흐름분석을 리용하여 항공음식전문회사제품을 재설계하시오.

13.11. (실례연구)트랜잭션분석을 리용하여 항공음식전문회사제품을 재설계하시오.

13.12. (실례연구) 부록 8과 부록 9의 상세설계들은 표형식으로 제시된다. 자신이 선택한 PDL(의사코드)을 리용하여 설계를 다시 제시하시오.

어느 표현이 더 좋은가? 그 대답과 리유를 주시오.

13.13. (쏘프트웨어공학독본) 교원은 문헌 [Stolper, 1999]의 복제본을 배포할것이다. 고전적파라다임을 리용하고 있는 어떤 개발기업체에 객체지향파라다임을 도입하는것과 관련한 당신의 견해는 무엇인가?

참 고 문 헌

[Beizer, 1990] B. BEIZER, *Software Testing Techniques*, 2nd ed., Van Nostrand Reinhold, New York, 1990.

[Bias, 1991] R. BIAS, "Walkthroughs: Efficient Collaborative Testing," *IEEE Software* **8** (September 1991), pp. 94–95.

[Binkley and Schach, 1996] A. B. BINKLEY AND S. R. SCHACH, "A Comparison of Sixteen Quality Metrics for Object-Oriented Design," *Information Processing Letters* **57** (No. 6, June 1996), pp. 271–75.

[Binkley and Schach, 1997] A. B. BINKLEY AND S. R. SCHACH, "Toward a Unified Approach to Object-Oriented Coupling," *Proceedings of the 35th Annual ACM Southeast Conference*, Murfreesboro, TN, April 2–4, 1997, pp. 91–97.

[Binkley and Schach, 1998] A. B. BINKLEY AND S. R. SCHACH, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," *Proceedings of the 20th International Conference on Software Engineering*, Kyoto, Japan, April 1988, pp. 542–55.

[Brandl, 1990] D. L. BRANDL, "Quality Measures in Design: Finding Problems before Coding," *ACM SIGSOFT Software Engineering Notes* **15** (January 1990), pp. 68–72.

[Brooks, 1986] F. P. BROOKS, JR., "No Silver Bullet," in: *Information Processing '86*, H.-J. Kugler (Editor), Elsevier North-Holland, New York, 1986. Reprinted in *IEEE Computer* **20** (April 1987), pp. 10–19.

[Chidamber and Kemerer, 1994] S. R. CHIDAMBER AND C. F. KEMERER, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering* **20** (June 1994), pp. 476–93.

[Coad and Yourdon, 1991b] P. COAD AND E. YOURDON, *Object-Oriented Design*, Yourdon Press, Englewood Cliffs, NJ, 1991.

[Cooling, 1997] J. E. COOLING, *Real-Time Software Systems: An Introduction*, Van Nostrand Reinhold, New York, 1997.

[Fagan, 1976] M. E. FAGAN, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15** (No. 3, 1976), pp. 182–211.

[Fagan, 1986] M. E. FAGAN, "Advances in Software Inspections," *IEEE Transactions on Software Engineering* **SE-12** (July 1986), pp. 744–51.

[Fichman and Kemerer, 1992] R. G. FICHMAN AND C. F. KEMERER, "Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique," *IEEE Computer* **25** (October 1992), pp. 22–39.

[Flanagan and Loukides, 1997] D. FLANAGAN AND M. LOUKIDES, *Java in a Nutshell: A Desktop Quick Reference*, 2nd ed., O'Reilly and Associates, Sebastopol, CA, 1997.

[Gane and Sarsen, 1979] C. GANE AND T. SARSEN, *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, Englewood Cliffs, NJ, 1979.

[Goldberg and Robson, 1989] A. GOLDBERG AND D. ROBSON, *Smalltalk-80: The Language*, Addison-Wesley, Reading, MA, 1989.

[Henry and Kafura, 1981] S. M. HENRY AND D. KAFURA, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering* **SE-7** (September 1981), pp. 510–18.

[Henry and Selig, 1990] S. HENRY AND C. SELIG, "Predicting Source-Code Complexity at the Design Stage," *IEEE Software* **7** (March 1990), pp. 36–44.

[Herbsleb and Grinter, 1999] J. D. HERBSLEB AND R. E. GRINTER, "Architectures, Coordination, and Distance: Conway's Law and Beyond," *IEEE Software* **16** (September/October 1999), pp. 63–70.

[Hoare, 1987] C. A. R. HOARE, "An Overview of Some Formal Methods for Program Design," *IEEE Computer* **20** (September 1987), pp. 85–91.

[ISO/IEC 8652, 1995] "Programming Language Ada: Language and Standard Libraries," ISO/IEC 8652, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Jackson, 1975] M. A. JACKSON, *Principles of Program Design,* Academic Press, New York, 1975.

[Jacobson, Booch, and Rumbaugh, 1999] J. RUMBAUGH, G. BOOCH, AND I. JACOBSON, *The Unified Software Development Process,* Addison Wesley, Reading, MA, 1999.

[Kelly and Sherif, 1992] J. C. KELLY AND J. S. SHERIF, "A Comparison of Four Design Methods for Real-Time Software Development," *Information and Software Technology* **34** (February 1992), pp. 74–82.

[Kitchenham, Pickard, and Linkman, 1990] B. A. KITCHENHAM, L. M. PICKARD, AND S. J. LINKMAN, "An Evaluation of Some Design Metrics," *Software Engineering Journal* **5** (January 1990), pp. 50–58.

[Kramer, 1994] J. KRAMER, "Distributed Software Engineering," *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 253–63.

[Levi and Agrawala, 1990] S.-T. LEVI AND A. K. AGRAWALA, *Real Time System Design,* McGraw-Hill, New York, 1990.

[McCabe, 1976] T. J. MCCABE, "A Complexity Measure," *IEEE Transactions on Software Engineering* **SE-2** (December 1976), pp. 308–20.

[Meyer, 1992b] B. MEYER, *Eiffel: The Language,* Prentice Hall, New York, 1992.

[Monarchi and Puhr, 1992] D. E. MONARCHI AND G. I. PUHR, "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM* **35** (September 1992), pp. 35–47.

[Orr, 1981] K. ORR, *Structured Requirements Definition,* Ken Orr and Associates, Inc., Topeka, KS, 1981.

[Shepperd, 1990] M. SHEPPERD, "Design Metrics: An Empirical Analysis," *Software Engineering Journal* **5** (January 1990), pp. 3–10.

[Shlaer and Mellor, 1992] S. SHLAER AND S. MELLOR, *Object Lifecycles: Modeling the World in States,* Yourdon Press, Englewood Cliffs, NJ, 1992.

[Silberschatz and Galvin, 1998] A. SILBERSCHATZ AND P. B. GALVIN, *Operating System Concepts,* 5th ed., Addison-Wesley, Reading, MA, 1998.

[Stankovic, 1997] J. A. STANKOVIC, "Real-Time and Embedded Systems," in: *The Computer Science and Engineering Handbook,* A. B. Tucker, Jr. (Editor-in-Chief), CRC Press, Boca Raton, FL, pp. 1709–24.

[Stolper, 1999] S. A. STOLPER, "Streamlined Design Approach Lands Mars Pathfinder," *IEEE Software* **16** (September/October 1999), pp. 52–62.

[Stroustrup, 1991] B. STROUSTRUP, *The C++ Programming Language,* 2nd ed., Addison-Wesley, Reading, MA, 1991.

[Walker, 1992] I. J. WALKER, "Requirements of an Object-Oriented Design Method," *Software Engineering Journal* **7** (March 1992), pp. 102–13.

[Ward and Mellor, 1985] P. T. WARD AND S. MELLOR, *Structured Development for Real-Time Systems,* Volumes 1, 2 and 3, Yourdon Press, New York, 1985.

[Warnier, 1976] J. D. WARNIER, *Logical Construction of Programs,* Van Nostrand Reinhold, New York, 1976.

[Wirfs-Brock, Wilkerson, and Wiener, 1990] R. WIRFS-BROCK, B. WILKERSON, AND L. WIENER, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, NJ, 1990.

[Yourdon and Constantine, 1979] E. YOURDON AND L. L. CONSTANTINE, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,* Prentice Hall, Englewood Cliffs, NJ, 1979.

[Zage and Zage, 1993] W. M. ZAGE AND D. M. ZAGE, "Evaluating Design Metrics on Large-Scale Software," *IEEE Software* **10** (July 1993), pp. 75–81.

제１４장. 실 현 단 계

실현은 상세설계를 코드로 넘기는 과정이다. 어떤 개별적인 사람에 의하여 실현이 수행될 때 그 과정은 비교적 잘 리해된다. 그러나 현재 대부분의 실천적인 제품들은 너무 방대하기때문에 한명의 프로그람작성자가 주어 진 시간내에 실현하기 어렵다. 그대신 제품은 그 제품의 각이한 부분들에 대하여 동시에 작업하는 개발팀에 의하여 실현된다. 이것을 협동프로그람작성(*programming-in-the-many*)이라고 부른다. 이 장에서는 협동프로그람작성과 관련된 문제들이 론의된다.

１４．１．프로그람작성언어의 선택

대부분의 경우에 실현을 위하여 어느 프로그람작성언어를 선택하겠는가 하는 문제는 단순하게 제기되지 않는다. 의뢰자가 어떤 제품이 Smalltalk로 작성되기를 바란다고 하자. 개발팀의 견해로서는 그 제품에 Smalltalk가 전혀 부합되지 않을수 있다. 이와 같은 견해는 의뢰자에게 적합치 않다. 그러면 개발기업체의 관리자측에게는 두개의 선택만이 존재하는데 그것은 그 제품을 Smalltalk로 실현하든가 그 주문을 거절하는것이다.

이와 류사하게 제품이 어떤 특정한 콤퓨터상에서 실현되여야 하며 또 그 콤퓨터에서 리용할수 있는 유일한 언어가 아쎔블리어라고 하면 거기에는 다른 선택이 없다. 그 콤퓨터상에서 동작하는 임의의 고급언어를 위한 콤파일러가 아직 작성되지 않았거나 관리자측이 규정된 콤퓨터를 위한 새로운 C++콤파일러를 사기 위하여 돈을 지불할 준비가 되여 있지 않는것으로 인하여 기타 다른 언어를 리용할수 없다면 프로그람작성언어의 선택에 관한 론의는 적당하지 않다.

보다 흥미 있는 문제는 다음과 같다. 어떤 계약서가 제품이 《가장 적당한》프로그람작성언어로 실현된다는것을 명시하고 있다. 어느 언어가 선택되여야 하는가? 이 문제에 답변하기 위하여 다음과 같은 대본을 고찰하자. QQQ회사는 25년이상 COBOL제품을 작성하여 왔다. 대부분의 중간급프로그람작성자로부터 쏘프트웨어의 부책임자에 이르기까지 QQQ회사의 200여명의 직원들이 COBOL전문지식을 가지고 있다. 무엇때문에 지구상에서 가장 적절한 프로그람작성언어가 COBOL이외의것으로 되여야 하는가? 어떤 새로운 언어 례하면 Java의 도입은 새로운 프로그람작성자들을 고용하든가 최소한 현재의 직원들이 집중적으로 재교육되여야 한다는것을 의미하게 된다. 관리자측은 Java교육에 자금과 노력을 모두 투자하면서 미래의 제품들이 Java로 작성되여야 한다고 결심할수도 있다. 그러나 현존 COBOL제품들도 모두 유지정비되여야 한다. 그렇게 되면 두가지 부류의 프로그람작성자들 즉 COBOL유지정비를 위한 프로그람작성자들과 새로운 응용제품들을 작성하기 위한 Java프로그람작성자들이 존재하게 될것이다. 부당하게도 유지정비는 거의 언제나 새로운 응용프로그람을 작성하는것보다 렬등하다고 간주되고 있다. 그러므로 COBOL프로그람작성자들은 분명히 불행한 처지에 놓이게 된다. 이러한 불행들은 Java프

로그람작성자들이 단기보수를 받기때문에 COBOL프로그람작성자들보다 더 많은 보수를 받게 된다는 사실로부터 생긴다. 비록 QQQ가 COBOL을 위한 뛰여 난 개발도구들을 가지고 있다 할지라도 적당한 Java도구들은 물론 Java콤파일러를 구입하여야 한다. 이 새로운 프로그람들을 실행하기 위하여 추가적인 하드웨어들을 구입하거나 빌려야 한다. 아마도 가장 심각한 문제는 QQQ가 COBOL에 대한 많은 인적 및 시간적전문지식을 축적하였으며 이러한 전문지식들은 어떤 점의 오유가 화면에 출현하였을 때 무엇을 하며 콤파일러의 급격한 변동을 어떻게 다루겠는가와 같이 수동적인 경험에 의해서만 얻어 질수 있다는것이다. 믿건대 《가장 적합한》프로그람작성언어는 오직 COBOL인것 같다. 임의의 다른 언어의 선택은 그에 드는 비용의 견지로 보나 직원들의 사기를 떨구어 불충분한 품질의 코드를 만들어 내게 된다는 견지로 보면 재정적인 자살행위로 될것이다.

알고 싶은 문제

다른 모든 프로그람작성언어를 합친것보다 더 많은 원천들이 COBOL로서 작성되였다. 본래 COBOL은 그것이 DoD의 제품이기때문에 가장 광범하게 리용되였다. 죽은 해군소장 그레이스 머레이 호퍼(Grace Murray Hopper)의 지도밑에 개발된 COBOL은 1960년 DoD에 의하여 인증되였다. 그이후부터 DoD는 장치가 COBOL콤파일러를 가지고 있지 않는 한 자료처리응용을 위하여 그 장치를 구입하지 않았다. DoD는 지금도 그러하지만 세계 최대의 콤퓨터하드웨어구입자였으며 1960년대에 DoD쏘프트웨어의 상당한 부분들이 자료처리를 목적으로 작성되였다. 그 결과 COBOL콤파일러는 사실상 매 콤퓨터에 대하여 선차적으로 작성되였다. 이와 같은 COBOL의 광범한 응용은 그에 대한 유일한 대용언어가 아셈블리어였던 시대에 COBOL이 세계적으로 가장 대중적인 프로그람작성언어로 되게 하였다.

C, C++, Java 4세대언어들과 같은 언어들은 새로운 응용들에서 의심할바없이 대중성이 증가되고 있다. 그러나 유지정비는 여전히 중요한 쏘프트웨어활동이며 이러한 유지정비는 현존 COBOL쏘프트웨어에서 수행되고 있다. 한마디로 말하면 DoD는 자기의 첫 프로그람작성언어 COBOL을 통하여 세계의 프로그람계에 자기의 이름을 남기였다.

COBOL이 대중성을 가지게 된 또 한가지 리유는 COBOL이 흔히 자료처리제품을 실현하기 위한 가장 좋은 언어로 되기때문이다. 특히 COBOL은 일반적으로 자료문제가 포함되는 경우에 선택하게 되는 언어이다. 재정장부들은 잔고가 없이 깨끗이 맞아 떨어져야 하며 따라서 둥그리기오유들이 끼여 드는것이 허락될수 없다. 그러므로 모든 계산들은 옹근수산수연산으로써 수행되여야 한다. COBOL은 대단히 큰 수들(즉 백만딸라)에 대한 산수연산을 지원한다. 그밖에 COBOL은 매우 작은 수들을 다룰수 있다. 은행규칙들은 적어도 소수점 4자리까지의 센트에 해당한 리자계산을 할것을 필요로 하는데 COBOL은 이러한 산수연산을 매우 쉽게 할수 있다. 마지막으로 COBOL은 가장 좋은 형식과 정렬방법 그리고 모든 3세대언어들이 가지고 있는 보고서생성프로그람들을 가지고 있다. 이러한 모든 리유로 하여 COBOL은 자료처리제품을 실현하기 위한 하나의 훌륭한 선택언어로 된다.

8.7.4에서 언급한바와 같이 지금 거의 완성단계에 이른 COBOL언어표준은 객체지향언어를 위한것이다. 이러한 표준은 틀림없이 COBOL의 대중성을 더욱 높여 주게 될것이다.

그러나 QQQ회사의 최신프로젝트에 가장 적합한 프로그람작성언어는 사실 COBOL이 아닌 어떤 다른 언어일수도 있다. COBOL이 세계적으로 가장 광범히 리용되는 프로그람작성언어로서의 지위를 차지하고 있음에도 불구하고(우의 《알고 싶은 문제》를 보시오.) COBOL은 다만 한가지 부류의 쏘프트웨어제품 즉 자료처리응용프로그람에 적합하다. 만일 QQQ회사가 이 부류밖의 쏘프트웨어제품들을 요구한다면 COBOL은 곧 자기의 인기를 잃게 된다. 실례로 QQQ가 인공지능기술을 리용하여 지식에 기초한 제품을 구성하려고 한다면 Lisp와 같은 AI언어가 리용될것이며 COBOL은 AI응용에 전혀 적합치 않다. 만일 대규모통신쏘프트웨어를 구성하게 되는 경우에는 QQQ가 전 세계에 분포된 수백여개의 지부들과 위성을 통해 결합될것이 요구되기때문에 Java와 같은 언어가 COBOL보다 훨씬 더 적합하다고 판명될것이다. 만일 QQQ가 조작체계, 콤파일러, 련결편집프로그람과 같은 체계쏘프트웨어를 작성하는 임무를 수행하게 된다면 COBOL은 거의 확정적으로 적합치 않다. 만일 QQQ가 국방계약에 종사하기로 결심한다면 관리자측은 COBOL이 실시간내장쏘프트웨어에 리용될수 없다는것을 곧 발견하게 될것이다.

어느 프로그람작성언어가 흔히 리용되겠는가에 관한 론점은 비용 대 리득분석(5.2)을 리용하여 결정될수 있다. 즉 관리자측은 COBOL을 리용하는것으로써 현재와 미래에 얻게 될 리득과 함께 COLOL실현에 드는 비용을 계산하여야 한다. 이러한 계산을 고찰하는 매 언어에 대하여 반복하여야 한다. 예상리득 즉 타산리익과 타산비용의 차가 가장 큰 언어가 적당한 실현언어로 된다. 어느 프로그람언어를 선택하겠는가를 결정하기 위한 또 한가지 방법은 위험분석을 리용하는것이다. 고찰하는 매 언어에 대하여 잠재적인 위험과 그것을 해소하는 방법들을 기록한 목록을 작성한다. 전반적인 위험이 가장 작은 언어가 선택되게 된다.

현재 쏘프트웨어개발기업체는 하나의 객체지향언어 즉 임의의 객체지향언어로 새로운 쏘프트웨어를 개발하라는 압력을 받고 있다. 이때 발생하는 문제는 다음과 같다. 적당한 객체지향언어는 무엇인가? 20년전에는 사실상 단 하나의 선택 즉 Smalltalk만이 가능하였다. 그러나 현재 대다수의 객체지향쏘프트웨어들은 C++로 작성되고 있다. 여기에는 여러가지 리유가 있다. 그 하나는 C++콤파일러의 광범한 응용성이다. 사실 C++콤파일러는 원천코드로부터 C에로 단순히 번역하며 그다음 C콤파일러를 호출한다. 그렇기때문에 C콤파일러를 가지고 있는 임의의 콤퓨터는 본질상 C++를 다룰수 있다.

그러나 C++가 대중화되고 있는 실제적인 원인은 그것이 C와 외적인 류사성이 있기때문이다. 유감스럽게도 많은 경영자들은 C++를 단순히 C의 상위모임으로서 간주하며 따라서 C를 알고 있는 임의의 프로그람작성자가 추가적인 부분들을 신속히 찾아 낼수 있다고 결론 짓는다. 실지로 문법적인 견지에서 C++는 본질상 C의 상위모임이다. 그러나 개념상 C++는 C와 전혀 다르다. C는 구조화파라다임의 제품이며 반면에 C++는 객체지향파라다임을 위한 제품이다. C++를 리용하는것은 객체지향기법이 리용되며 제품이 모듈이 아니라 객체들과 클라스들로 조직되는 경우에만 합리적이다.

그러므로 어떤 개발기업체가 C++를 받아 들이기에 앞서 련관된 쏘프트웨어전문가들을 객체지향파라다임으로 숙련시키는것이 본질적이다. 제7장의 정보들을 가르쳐 주는것이 특별히 중요하다. 객체지향파라다임이 쏘프트웨어를 개발하는 다른 방법이며 명백한 차이가 무엇인가를 모든 성원들 특히 관리자측에 명백하지 않는 한에서는 C가

아니라 C++로 작성된 코드이지만 구조화파라다임이 계속 리용되게 될것이다. 개발기업체가 C로부터 C++로 전환한 결과에 실망하게 되는 경우에 한가지 중요한 인자는 객체지향파라다임에 대한 교육을 받지 못한것이다. 가령 어떤 개발기업체가 Java를 받아 들이기로 결심하였다고 하자. 이 경우에 구조화파라다임으로부터 객체지향파라다임에로 점차적으로 이행하는것은 불가능하다. Java는 순수한 객체지향프로그람작성언어이며 구조화파라다임의 기능들과 절차들은 지원하지 않는다. C++와 같은 혼성객체지향언어와는 달리 Java프로그람작성자들은 처음부터 객체지향파라다임(오직 객체지향파라다임만)을 리용하여야 한다. 하나의 파라다임으로부터 다른 파라다임에로 급격히 이행하여야 할 필요성으로부터 개발과정이 C++나 OO-COBOL과 같은 혼성객체지향언어에로 전환하려고 할 때보다 Java(또는 Smalltalk와 같은 순수한 객체지향언어)를 받아들이려 하는 경우 교육과 숙련은 한층 더 중요한 문제로 나선다. 4세대언어로 실현하면 어떤가? 이 론의는 다음절에서 진행된다.

14. 2. 4세대언어

최초의 콤퓨터들은 해석기도 콤파일러도 없었다. 이 콤퓨터들은 배선판으로 장치화하거나 스위치를 설정하는것으로서 2진코드로 프로그람작성되였다. 이와 같은 2진기계코드가 1세대언어였다. 2세대언어들은 1940년대 말과 1950년대 초에 개발된 아쎔블리어들이다. 프로그람들을 2진코드로 작성하는것 대신에 명령들은

mov $17, next

와 같은 기호적인 표식으로 표현되였다. 일반적으로 매개 아쎔블리어명령들은 하나의 기계코드명령으로 번역된다. 그러므로 비록 아쎔블리어가 기계코드보다 작성하기 쉽고 유지정비를 진행하는 프로그람작성자가 리해하기 쉽다고 하여도 아쎔블리어원천코드는 기계코드와 같은 길이를 가지였다.

C, C++, Pascal 또는 Java와 같은 3세대언어(또는 고급언어)의 리면에 있는 착상은 고급언어의 한개 명령문이 5 또는 10개의 기계코드명령으로 번역된다는것이다(이것은 또 하나의 추상화실례이다. 7.4.1을 보시오.). 결국 고급언어코드는 그와 등가인 아쎔블리어코드보다 상당히 짧다. 고급언어코드는 또한 리해하기 쉬우며 그로부터 아쎔블리어코드보다 유지정비하기가 쉽다. 고급언어들은 그와 등가적인 아쎔블리어코드만큼 그렇게 효률적이 아닐수도 있지만 이것은 유지정비에서의 편리성에 비해 볼 때 큰 문제가 아니다. 이러한 개념은 1970년대 말에 더욱 확인되였다. 4세대언어(4GL)설계의 한가지 중요한 목적은 매개의 4GL명령문이 30 지어는 50개의 기계코드명령과 등가로 되게 하는것이다. Focus 또는 Natural과 같은 4GL로 작성된 제품들은 원천길이가 보다 짧으며 때문에 개발이 보다 빠르고 유지정비하기도 더 쉽다.

기계코드로 프로그람을 작성하는것은 어렵다. 아쎔블리어로 프로그람을 작성하는것은 약간 더 쉬우며 고급언어를 리용하는것은 훨씬 더 쉽다. 4GL의 두번째로 중요한 설계목적은 프로그람작성에서의 편리성이다. 특히 대부분의 4GL들은 비절차적이다(이것을 명백히 하려면 다음의 《알고 싶은 문제》를 보시오.). 실례로 그림 14-1에 보여 준

명령을 고찰하자. 이 비절차적명령을 절차적으로 실행될수 있는 기계코드명령렬로서 번역하는것은 4GL콤파일러의 책임이다.

> ### 알고 싶은 문제
>
> 몇년전에 나는 뉴욕의 그랜드쎈트럴(Grand Central)역전밖에서 택시를 불러 세우고 운전수에게 《링컨쎈터로 갑시다.》라고 말하였다. 이것은 하나의 비절차적요구이다. 왜냐하면 내가 희망하는 결과를 말하였지만 그 결과를 어떻게 달성하겠는가 하는 결심을 운전수에게 맡겼기때문이다. 나는 그 운전수가 이 나라에서 사는지 2개월도 안되는 중앙유럽에서 온 이민자이며 사실 이 도시를 전혀 모르고 있다는것을 알게 되였다. 그리하여 나는 재빨리 비절차적요구를 다음과 같은 절차적요구로 바꾸었다.
>
> 《곧추, 곧추 갑시다. 다음번 신호등에서 오른쪽으로 꺾으시오. 오른쪽입니다. 오른쪽 바로 여기서, 예, 오른쪽입니다. 이제는 곧추 갑시다. 천천히 갑시다. 천천히!》 이렇게 링컨쎈터에 도착할 때까지 계속 하였다.

for every surveyor
    if rating **is** excellent
        add 6500 **to** salary

그림 14-1. 비절차적인 4세대언어

4GL으로 전환한 개발기업체들이 성공한 실례는 많다. 이전에 COBOL을 리용한 몇개의 개발기업체들은 4GL을 리용함으로써 실지로 생산성을 10배 늘이였다고 보고하였다. 많은 개발기업체들이 4GL을 리용함으로써 생산성이 실지로 증가하였다는것을 발견하였지만 그렇게 극적인것은 아니다. 다른 기업체들은 4GL을 시도하였으나 그 결과에 크게 실망하였다.

이와 같은 불일치 원인은 하나의 4GL이 모든 제품들에 적당한것 같지 않다는것이다. 반대로 특정한 제품에 대하여 정확한 4GL을 선택하는것이 중요하다. 실례로 Playtex 회사는 IBM의 응용개발프로그람(*Application Development Facility*; ADF)을 리용하고 COBOL보다 80대 1의 생산성증대를 보고하였다.

이런 뚜렷한 결과에도 불구하고 Playtex는 그이후에도 관리자측이 응용프로그람개발기능(ADF)에 덜 적합하다고 생각하는 제품들에 COBOL을 리용하였다[Martin, 1985].

이러한 불일치에 대한 또 하나의 리유는 대부분의 4GL들이 강력한 CASE작업대들과 환경들로 지원되고 있다는것이다(5.5).

CASE작업대들과 환경들은 장점과 약점을 모두 가질수 있다. 2.11에서 설명한바와 같이 낮은 성숙단계에 있는 개발기업체에서는 대규모CASE를 도입하지 않는것이 더 좋다. 그 원인은 CASE작업대나 환경의 목적이 쏘프트웨어개발공정을 지원하는것이기때문이다. 1준위에 있는 개발기업체는 쏘프트웨어개발공정을 가지지 못한다. 만일 이 시점에서 CASE가 어떤 4GL에로 이행하기 위한 수단으로써 도입된다면 이것은 임의의 공정에 준

비되여 있지 않는 개발기업체에 어떤 공정을 떠맡기게 될것이다. 그 결말은 일반적으로 불만족스러우며 또 비참하게 될수 있다. 사실 이미 보고된 많은 4GL실패들은 4GL 그자체가 아니라 련관된 CASE환경의 영향에 기인될수 있다.

4GL에 대한 43개 개발기업체들의 립장에 대하여 문헌 [Guimaraes, 1985]에서 보고되였다. 4GL의 리용은 사용자가 개발기업체의 자료기지에서 추출한 정보들을 요구할 때 자료처리부서가 보다 신속히 응답할수 있게 하기때문에 사용자오유를 줄인다. 그러나 여기에는 여러가지 문제점들도 있다. 일부 4GL들은 오랜 응답시간으로 하여 느리고 효과적이 아니라는것이 증명되였다. 하나의 제품은 기껏하여 12명의 협동하는 사용자들을 지원하면서 IBM 4331주콤퓨터상에서 CPU주기의 60%를 소비하였다. 전체적으로 4GL을 3년동안 리용한 28개의 개발기업체들이 리득이 비용을 릉가하였다는것을 인식하였다.

어느 한 4GL도 쏘프트웨어시장을 독점하지 못하고 있다. 반면에 수백여개의 4GL이 존재하고 있다. DB2, Oracle, PowerBuilder을 비롯한 일부 언어들이 상당한 규모의 사용자그룹을 형성하고 있다. 4GL의 광범한 증대는 4GL을 정확히 선택할 때 심중한 주의를 돌려야 한다는것을 말해 주고 있다. 물론 몇개의 개발기업체들은 하나이상의 4GL을 지원할수 있는 여유가 있다. 일단 하나의 4GL이 선택되여 리용되면 개발기업체는 련이은 제품들에 대하여 그 4GL을 리용하든가 4GL을 도입하기전에 리용한 언어에 의거하든가 하여야 한다.

생산성에서 잠재적인 리득이 있음에도 불구하고 4GL을 잘못 리용할 때에 잠재적인 위험성이 존재한다. 많은 개발기업체들은 현재 개발하는 제품들에 대한 큰 예비를 가지고 있으며 또 수행해야 할 유지정비와 관련한 많은 과제들을 가지고 있다. 대다수의 4GL의 설계목적은 말단사용자프로그람작성(*end-user programming*) 즉 그 제품을 리용할 사람들이 프로그람을 작성하게 하는것이다. 실례로 4GL이 출현하기전에 보험회사의 투자관리자는 자료처리관리자에게 채권명세표를 고려하여 어떤 정보를 현시하여 주는 제품을 요청하였을것이다. 그다음 투자경영자는 1년동안 기다리든가 자료처리그룹이 시간을 얻어 내여 그 제품을 개발하도록 할것이다. 4GL은 이전에 프로그람작성을 해보지 않은 투자경영자가 남의 도움을 받지 않고 희망하는 제품을 작성할수 있을만큼 그렇게 단순하게 리용할수 있을것을 기대하였다. 말단사용자프로그람작성은 전문가들에게 현존하는 제품들에 대한 유지정비를 맡기고 제품개발에 대한 재고량을 줄이는것을 도울 작정이였다.

실천적으로 말단사용자프로그람작성은 위험을 동반할수 있다. 먼저 모든 제품개발이 전문가들에 의하여 수행되는 정황을 고찰하자. 콤퓨터전문가들은 콤퓨터의 출력을 믿지 않는데 습관되여 있다. 결국 제품개발기간에는 모든 출력의 1%미만이 정확할수 있다. 한편 제품에 오유가 없을 때까지는 그 어떤 제품도 사용자에게 배포되지 말아야 하기때문에 사용자는 모든 콤퓨터출력을 믿게 되여 있다. 이제는 말단 사용자프로그람작성이 장려되는 정황을 고찰하자. 프로그람작성에 경험이 없는 사용자가 사용하기 쉬운 비절차적 4세대언어로 코드를 작성할 때 사용자가 출력을 믿는것이 자연스러운 경향이다. 결국 몇해동안 그 사용자는 콤퓨터의 출력을 믿도록 명령 받아 왔다. 결과 많은 업무결정들은 매우 부정확한 말단사용자코드로 생성된 자료에 의거하여 왔다. 일부 경우에 어떤 4GL의 사용자친절성은 재정적인 파산을 초래하였다.

또 한가지 잠재적인 위험요소는 일부 개발기업체들에서 사용자들이 개발기업체의 자

료기지를 갱신하는 4GL제품들을 작성하도록 하려는 경향이다. 사용자에 의하여 생겨 난 프로그람작성오유는 종국적으로 전체 자료기지의 와전을 초래할수 있다. 경험은 명백하다. 경험이 없거나 숙련되지 못한 사용자들이 프로그람을 작성하는것은 비록 극히 치명적이 아니라고 해도 회사에 재정적손해를 끼치게 될것이다. 4GL의 최종선택은 관리자측이 하게 된다.

이러한 결심을 할 때 관리자측은 4GL리용에서의 많은 성공실례들은 참고하여야 한다. 동시에 관리자측은 부적당한 4GL의 리용, CASE환경의 신속도입 또는 개발공정의 불충분한 경영으로 인하여 초래된 실태들을 세밀히 분석하여야 한다. 실례로 실패의 공통적인 원인은 관계자료기지리론[Date, 1999]을 비롯하여 4GL의 모든 측면들에 대하여 개발팀이 철저히 파악하는것을 무시하고 있는것이다. 경영자는 특정한 응용령역에서의 성공과 실패를 모두 연구하고 과거의 오유로부터 교훈을 찾아야 한다. 정확한 4GL을 선택하는것은 커다란 성공과 비참한 실패의 갈림길을 의미할수 있다.

실현언어를 결정하였기때문에 다음의 론점은 쏘프트웨어공학의 원리들이 어떻게 보다 품질이 좋은 코드작성에로 이끌어 갈수 있겠는가 하는 문제이다.

14．3．훌륭한 프로그람작성실천

훌륭한 코드작성품격과 관련한 대부분의 권고들은 특정한 언어에 국한되여 있다. 실례를 들면 COBOL88준위입력 또는 Lisp에서의 괄호의 리용과 관련한 제안들은 Java로 제품을 실현하는 프로그람작성자들에게는 그리 흥미가 없다. 실현에 열중하고 있는 독자들에게는 헨리 레가드(Henry Ledgard)가 쓴 책과 같은 여러 책들중에서 어느 하나를 읽을것이 요구되는데 그러한 책들에서는 그 제품이 실현되고 있는 특정한 언어에서의 훌륭한 프로그람작성습관들에 대하여 설명하여 준다. 아래에서 언어에 의존하지 않는 훌륭한 프로그람작성습관에 관하여 몇가지 권고한다.

일관하고 의미 있는 변수이름의 리용 제1장에서 서술한것처럼 쏘프트웨어운영비의 평균 3분의 2는 유지정비에 돌려 지고 있다. 이것은 어떤 모듈을 개발하고 있는 프로그람작성자는 그 모듈에서 작업하는 많은 사람들가운데서 첫 사람에 지나지 않는다는것을 의미한다. 어떤 프로그람작성자에게 있어서 변수들에 자기에게만 의미가 있는 이름을 주는것은 비생산적이다. 여기서 쏘프트웨어공학의 범위내에서 용어 《의미 있는》은 《유지정비를 진행하는 미래의 프로그람작성자의 견지에서 《의미 있는》을 의미하고 있다. 다음의 《알고 싶은 문제》에서 이 점에 대하여 설명한다.

의미 있는 변수이름을 리용하는것외에 변수이름들이 일치하는것이 마찬가지로 중요하다. 실례로 하나의 모듈안에서 다음의 4개 변수 즉 averageFreq, frequencyMaximum, minFr, frqncyTotl이 선언되였다고 하자. 코드를 리해하려고 하는 유지정비프로그람작성자는 freq, frequency, fr, frqncy가 모두 같은것과 관계되여 있는가를 알아야 한다. 만일 같은것과 관계되여 있다면 식별단어가 리용되여야 한다. 비록 freq 또는 frqncy는 부분적으로 마음에 들지만 fr는 마음에 들지 않으므로 오히려 frequency를 리용한다. 그러나 만일 하나 또는 그이상의 변수이름들이 다른 량과 관련되여 있다면 rate와 같이 전혀 다른 이름

이 리용되여야 한다. 거꾸로 동일한 개념을 지적하기 위하여 두개의 각이한 이름을 리용하지 말아야 한다. 실례로 average와 mean은 모두 같은 프로그람내에서 리용되지 말아야 한다.

알고 싶은 문제

1970년대 말에 남아프리카의 요한네스부르그에는 두개의 팀으로 구성된 하나의 작은 쏘프트웨어개발기업체가 있었다. 팀 A는 모잠비크에서 온 이주민들로 구성되였는데 그들은 뽀르뚜갈태생이였고 모국어는 뽀르뚜갈어였다. 그들이 작성하는 코드는 훌륭하였다. 변수이름들은 의미가 있었지만 유감스럽게도 뽀르뚜갈어로 말하는 사람들에게만 그러하였다. 팀 B는 모국어가 헤브라이어인 이스라엘이주민들로 조직되였다. 그들이 작성하는 코드도 훌륭하였고 변수이름들도 의미가 있었지만 헤브라이어로 말하는 사람에게만 그러하였다.

어느 날 팀 A가 팀책임자와 함께 집단적으로 사직하였다. 팀 B는 팀 A가 작성한 코드의 어느 하나도 전혀 유지정비할수 없었다. 왜냐하면 그들은 뽀르뚜갈어로 말하지 못하였기때문이다. 뽀르뚜갈어로 말하는 사람인 경우에만 의미가 있는 변수이름들은 헤브라이어와 영어로 말할수 있는 이스라엘인들에게는 리해될수 없었다. 쏘프트웨어개발기업체 책임자는 팀 A를 대신할 뽀르뚜갈어로 말할수 있는 프로그람작성자들을 충분히 고용할수 없었고 회사는 자기들의 원천코드를 더는 유지정비할수 없게 된것으로 하여 불만을 품은 손님들의 법적배상압력을 받고 곧 파산되게 되였다.

이 정황은 일찌기 피할수 있는것이였다. 회사사장은 시작부터 모든 변수이름을 모든 남아프리카 콤퓨터전문가들이 리해할수 있는 언어인 영어로 작성하도록 요구했어야 하였다. 그러면 변수이름들은 유지정비를 진행하는 모든 프로그람작성자들에게 의미가 있었을 것이였다.

일치성의 두번째 측면은 변수이름에서 요소들의 순서이다. 실례로 만일 한 변수가 frequencyMaximum으로 이름 지어 졌다면 minimumFrequency라는 이름은 혼동될것이다. 그것은 FrequencyMinimum으로 이름 지어 져야 한다. 앞으로의 유지정비프로그람작성자에게 코드가 명백하고 애매성이 없도록 하기 위하여 앞에서 렬거한 4개의 변수들은 각각 다음과 같이 이름 지어야 한다. 즉 frequencyAverage, frequencyMaximum, frequencyMinimum, frequencyTotal이다. 이 대신에 요소 frequency를 4개의 변수이름의 끝에 오게 할수도 있다. 즉 변수이름은 averageFrequency, maximumFrequency, minimumFrequency, totalFrequency로 할수 있다. 이 두 변수이름모임중에서 어느것을 선택하는가는 문제가 아니다. 중요한것은 우의 어느 한 모임에서 모든 이름들을 선택하는것이다.

코드를 보다 쉽게 리해할수 있게 하는 여러가지 서로 다른 이름짓기습관들이 제시되였다. 그 착상은 어떤 변수의 이름에 류형정보를 병합하는것이다. 실례로 들면 ptrChTmp는 어떤 문자(Ch)에 대한 지적자형(ptr)림시변수(Tmp)를 의미하고 있다. 이와 같은 가장 널리 알려 진 이름짓기방법은 마쟈르식이름짓기습관이다[Klunder, 1988](만일 왜 마쟈르식이라고 부르는가를 알고 싶다면 다음의 《알고 싶은 문제》를 보시오.). 이러한 여러가지

방법들이 가지고 있는 한가지 결함은 대방이 변수이름들을 발음할수 없을 때 코드검토의 효과성(14.9)이 감소될수 있다는것이다. 변수이름들을 한글자씩 발음하여야 하는것은 매우 시끄러운 일이다.

알고 싶은 문제

용어 마쟈르식이름짓기습관(*Hungarian Naming Conventions)*에 대하여서는 두가지로 설명할수 있다. 첫째는 이 습관이 마쟈르에서 출생한 챨스 시모니(Charles Simonyi)에 의해 발명되였다는것이다. 둘째는 일반적으로 인정되고 있는것인데 초학도에게 있어서 이 습관에 따르는 변수이름들로써 프로그람을 작성하는것은 마쟈르어처럼 읽기가 쉽다는것이다. 그러나 이 습관을 리용하는 기업체(마이크로쏘프트와 같은)들은 이 습관이 마쟈르식이름짓기관습에 경험이 있는 사람들에게서 코드가독성을 증가시킨다는것을 주장하고 있다.

자기식으로 문서화한 코드작성문제 프로그람작성자들에게 도대체 무엇때문에 그들이 작성한 코드에 아무런 설명문도 없는가고 물으면 프로그람작성자는 자주 《나는 자기식으로 문서화한 코드를 작성한다.》고 자랑스럽게 답변한다. 이것은 그들이 리용하는 변수이름들이 아주 주의 깊게 선택되고 그들이 작성한 코드들이 그 어떤 설명문도 필요 없을 정도로 섬세하게 작성된다는것을 의미한다. 자기 식으로 문서화한 코드는 존재하기는 하지만 극히 드물다. 그대신 일반적으로 쓰이는 방법은 모듈이 작성될 때 프로그람작성자가 코드에서의 모든 미묘한 차이를 판별하는것이다. 프로그람작성자가 매 모듈에 같은 형식을 리용한다면 5년내에서는 코드들이 원래의 프로그람작성자의 모든 측면에서 여전히 명백할것이라는것은 상상할수 있는 일이다.

유감스럽게도 이것은 잘못된 상상이다. 중요한 문제는 쏘프트웨어품질보증그룹으로부터 시작하여 수많은 유지정비프로그람작성자들에 이르기까지 그 코드를 읽는 다른 모든 프로그람작성자들에게 모듈이 쉽게 명백하게 리해될수 있는가 하는것이다. 문제는 경험이 없는 프로그람작성자들에게 유지정비과업을 맡기고 그들을 철저히 관리하지 않는 잘못된 실천경우가 있게 되는것과 관련하여 더욱 심각하게 제기된다. 모듈들에 대한 문서화되지 못한 코드는 경험 있는 프로그람작성자에게도 부분적으로만 리해될수 있다. 더욱 한심한 경우는 유지정비프로그람작성자가 경험이 없을 때이다.

있을수 있는 문제들의 종류를 보기 위하여 변수 xCoordinateOfPositionOfRobotArm을 고찰하자. 이와 같은 변수이름은 의심할바없이 단어의 모든 의미에서 자기 식의 문서화를 실현하고 있지만 많은 프로그람작성자들은 31개 문자로 된 변수이름을 리용하려 하지 않으며 특히 그 변수가 자주 리용된다면 더욱 그러하다. 그대신 보다 짧은 이름 례하면 xCoord를 리용한다. 그 리면에 있는 원인은 만일 총적인 모듈이 로보트팔의 움직임을 처리한다면 xCoord는 그 로보트팔위치의 X자리표와만 련관될수 있다는것이다. 비록 이 변수는 개발공정의 범위내에서는 리치에 맞지만 유지정비를 위하여서는 필수적으로 옳은것이 아니다.

유지정비프로그람작성자에게 전반적제품에 대한 충분한 지식이 없어서 이 모듈내에서 xCoord는 로보트의 팔과 관련된다는것을 리해할수 없을수도 있고 그 모듈의 작업을 리해하는데 필요한 문서가 없을수도 없다. 이러한 종류의 문제를 극복하기 위한 한가지 방법은 매 변수의 이름을 프로그람의 서두 즉 서두설명문(*prologue comments*)에서 설명하도록 요구하는것이다. 만일 이 규칙을 따르게 된다면 유지정비프로그람작성자는 변수 xCoord가 로보트팔위치의 x자리표를 위하여 리용된다는것을 인차 리해하게 될것이다.

서두설명문은 매개의 단독 모듈들에서 필수적인것이며 매 모듈의 시작에서 제공되여야 한다. 최소한의 정보는 다음과 같다. 즉 모듈의 이름, 모듈의 사명에 대한 간단한 설명, 프로그람작성자의 이름, 모듈이 작성된 날자, 모듈이 허가된 날자, 모듈의 인수들, 자모순으로 정렬된 변수목록, 모듈이 호출하는 파일이름, 이 모듈에 의하여 변경되는 파일들의 이름, 오유처리능력, 이후의 회귀시험를 위해 리용되는 시험자료를 포함하고 있는 파일이름, 변경목록, 변경의 날자와 변경자, 알려 진 오유들이다.

비록 어떤 모듈이 명백하게 작성된다고 하여도 그 모듈의 사명과 그것이 어떻게 실현되는가를 리해하기 위하여 매개 행을 읽어야 한다면 그것은 타당치 않다. 서두설명문은 다른 사람들이 중요한 점들을 쉽게 리해할수 있게 한다. SQA그룹성원 또는 어떤 특정한 모듈을 변경하는 유지정비전문가들만이 그 모듈의 매행을 읽게 하여야 한다.

서두설명문외에 직접삽입설명문(*inline comment*)을 코드에 삽입하여 유지정비전문가들이 그 코드를 리해하는것을 도울수 있게 하여야 한다. 코드가 명확치 않은 방식으로 작성되거나 해당 언어의 아주 리해하기 힘든 측면들을 리용하는 경우에만 직접삽입설명문을 리용하여야 한다. 반대로 혼돈을 일으키는 코드들은 어떤 보다 명백한 방법으로 작성되여야 한다. 직접삽입설명문은 유지정비프로그람작성자들을 돕기 위한 수단이며 불충분한 프로그람작성실천을 추동하거나 변명하는데 리용하여서는 안된다.

파라메터들의 리용 절대상수 즉 값이 절대로 변하지 않는 변수들은 매우 적다. 실례로 위성사진들은 하와이의 펄 하버(Pearl Harbor)의 위도와 경도를 병합하고 있는 해저항해체계에서 변경되여 펄 하버의 정확한 위치에 관하여 보다 정확한 기하학적자료들을 반영하도록 되여 있다. 또 다른 실례를 들어 보면 판매세는 절대상수가 아니다. 립법자들은 시시각각 판매세를 변경시키려고 시도한다. 가령 판매세률이 현재 6.0%라고 하자. 만일 값 6.0이 어떤 제품의 여러가지 모듈안에 고정되여 있다면 그 제품을 변경시키는것은 하나의 기본과제로 된다. 이때 《상수》 6.0의 하나 또는 두개 실례의 가능한 결과를 무시하고 어떤 상관 없는 6.0을 실수로 변경시키게 된다.

한가지 좋은 해결대책은 다음과 같은 C++선언

```
const float salesTaxRate = 6.0;
```

이나 Java선언

```
public static final float salesTaxRate = (float) 6.0;
```

을 리용하는것이다. 그러면 판매세률값이 요구될 때마다 상수 salesTaxRate가 리용되고 수 6.0은 리용되지 않게 된다. 만일 판매세률이 변화된다면 salesTaxRate의 값을 포함하고 있는 행만 편집기로 변경시키면 된다. 더 좋기는 판매세의 값은 실행서두에서 어떤

파라메터파일로부터 읽어 들여야 한다. 이와 같은 피상적인 상수들은 파라메터로서 취급되여야 한다. 이렇게 되면 만일 어떤 값이 임의의 원인으로 하여 변경되여야 한다면 이 변경들은 신속히, 효과적으로 실현될수 있다.

가독성을 증대시키기 위한 코드배치 어떤 모듈을 읽기 쉽게 하는것은 비교적 간단하다. 실례로 비록 대부분의 프로그람작성언어들에서 한 렬에 한개이상의 명령문들을 쓸수 있다고 하여도 한 행에 한개이상 명령문을 쓰지 않는 방법이다. 아마도 들여쓰기방법은 가독성을 증대시키기 위한 가장 중요한 기법으로 될것 같다. 만일 들여쓰기를 리용하여 코드를 리해하기 쉽게 하지 않았더라면 제7장의 코드실례들은 얼마나 리해하기 어렵겠는가를 상상해 보라. C++ 또는 Java에서 들여쓰기는 대응하는 {...} 쌍들을 련결시키는데 리용될수 있다. 들여쓰기는 또한 어느 명령문들이 주어 진 블로크에 속하는가를 보여 준다. 사실 들여쓰기는 인간적존재를 무시할수 있을 정도로 매우 중요하다. 그대신 5.6에서 서술한바와 같이 CASE도구들을 리용하여 들여쓰기가 정확히 수행된다는것을 확인하여야 한다.

또 한가지 유용한 수단은 공백행들이다. 모듈들은 공백행들로 분리되여야 한다. 공백행들로 큰 코드블로크들을 갈라 놓는것은 도움이 된다. 큰 《흰 공백공간》은 코드를 보다 읽기 쉽고 리해하기 쉽게 한다.

함유된 if 명령문 다음의 실례를 고찰하자. 그림 14-2에 보여 준것처럼 어떤 지도가 두개의 정방형으로 구성되여 있다. 지구면우의 어떤 점이 지도정방형 1(*map square 1*) 또는 지도정방형 2(*map square 2*)에 속하는가를 결정하기 위한 코드를 작성하는것이 필요하다. 그림 14-3의 코드는 리해하기 어렵게 형식화되였다. 적당하게 형식화된 코드를 그림 14-4에 제시한다. 그럼에도 불구하고 **if-if**와 **if-else-if**구조들의 결합이 너무 복잡하여 코드토막들의 정확성여부를 검열하기 어렵다. 그림 14-5에서는 이것을 수정하고 있다.

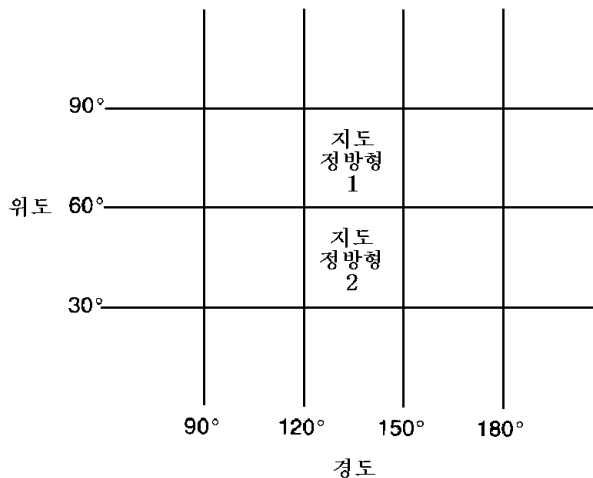



그림 14-2. 지도좌표

if-if구조를 포함하고 있는 복잡한 코드에 부닥칠 때 간단히 하기 위한 한가지 방법은

if-if결합

if ＜*condition* 1＞
　　if ＜*condition* 2＞

가 비록 ＜*condition* 1＞이 성립하지 않는다고 해도 ＜*condition* 2＞는 정의된다는 조건하에서 단독조건

if ＜*condition* 1＞ **and** ＜*condition* 2＞

와 등가이라는 사실을 리용하는것이다.

```
if (latitude>30 && longitude>120)  {if (latitude<=60 && longitude<=150)
mapSquareNo=1; else if (latitude<=90 && longitude<=150)  mapSquareNo=2
else print "Not on the map"; } else print "Not on the map";
```

그림 14-3. 잘못 형식화되여 있는 함유 **if** 명령

```
if (latitude>30 && longitude>120)
{
   if (latitude<=60 && longitude<=150)
         mapSquareNo=1;
   else if (latitude<=90 && longitude<=150)
         mapSquareNo=2
   else print "Not on the map";
}
else
      print "Not on the map";
```

그림 14-4. 잘 형식화되였으나 잘못 구조화되여 있는 함유 **if** 명령

```
if (longitude >120 && longitude><=150 && latitude>30 && latitude <=60)
        mapSquareNo=1;
else if (longitude >120 && longitude<=150&& latitude>60 && latitude <=90)
        mapSquareNo=2
else print "Not on the map";
```

그림 14-5. 허용할수 있는 함유 **if** 명령

실례로 ＜*condition* 1＞은 어떤 지적자가 령이 아니라는것을 검사할수 있으며 만일에 령이 아니라면 ＜*condition* 2＞는 그 지적자를 리용할수 있다(이 문제는 Java나 C++에서는 발생하지 않는다. &&연산자는 만일 ＜*condition* 1＞이 실패하면 ＜*condition* 2＞가 평

가되지 않는다는것으로서 정의된다.).

if-if구조와 관련된 또 한가지 문제는 함유 **if**명령문들이 너무 길어서 읽기 어려운 코드를 생성할수 있다는것이다. 경험적인 규칙으로서 3이상 깊이로 함유된 **if**명령문들은 좋지 못한 프로그람작성실례로서 피하여야 한다.

14. 4. 코드작성표준

코드작성표준은 하나의 행운으로 될수도 있고 하나의 재앙으로 될수도 있다. 7.2에서는 일치적인 응집도를 가진 모듈들은 일반적으로 《매 모듈들은 35～50개의 실행가능한 명령문들로 구성될것이다.》와 같은 규칙들의 결과로서 생성된다. 이와 같은 독단적인 방식으로 규칙을 서술하는것 대신에 다음과 같은 형식화를 리용하는것이 더 좋다. 《프로그람작성자들은 35개보다 적든가 50개보다 많은 실행가능한 명령문들을 가진 모듈들을 작성하기전에 관리자에게 물어 보아야 한다.》 요점은 가능한 모든 정황에 응용될수 있는 코드작성표준은 없다는것이다.

상부로부터 강요되는 코드작성표준들은 무시되는 경향이 있다. 이미 언급한바와 같이 한가지 쓸모 있는 경험적규칙은 **if**명령문이 3이상 깊이로 함유되지 말아야 한다는것이다. 만일 프로그람작성자들에게 **if**명령문을 너무 깊이 함유하여 읽을수 없는 코드실례들을 제시하여 준다면 프로그람작성자들은 이와 같은 규칙을 따르듯도 하다. 그러나 아무런 론의나 설명도 없이 강요되는 코드작성규칙들은 프로그람작성자들이 지키지 않을수 있다. 더우기 이와 같은 기준들은 프로그람작성자들과 경영자들사이의 마찰을 초래하게 될수도 있다. 더우기 어떤 코드작성표준은 기계로 검사될수 없는 한 그것은 SQA그룹에게 있어서 많은 시간을 랑비하게 하든가 프로그람작성자들과 SQA그룹에 의하여 단순히 무시되게 된다. 한편 다음과 같은 규칙을 고찰하자.

- **if**명령문의 함유는 개발팀책임자의 사전동의가 없는 한 3개 깊이를 초과하지 말아야 한다.
- 모듈들은 개발팀책임자의 사전동의가 없는한 35～50개 명령문들로 구성되여야 한다.
- **goto**명령문의 리용은 피해야 한다. 그러나 개발팀책임자의 사전동의가 있으면 앞방향 **goto**명령문을 오유처리를 위하여 리용할수 있다.

이와 같은 규칙들은 기준에서 리탈시키는 허가와 관계되는 자료들을 포착하기 위하여 어떤 기구가 설치된다는 조건하에서 기계적으로 검사될수 있다.

코드작성표준의 목적은 유지정비를 보다 쉽게 하는것이다. 그러나 만일 어떤 기준이 쏘프트웨어개발자들의 생존을 어렵게 한다면 비록 프로젝트의 실현도중이라고 하여도 그 기준을 변경하여야 한다. 지나치게 제한적인 코드작성표준들은 비생산적이며 만일 프로그람작성자들이 이와 같은 틀안에서 쏘프트웨어를 개발하여야 한다면 쏘프트웨어생산의 질이 부득이 손해를 입게 된다. 한편 **if**명령문의 함유, 모듈크기, **goto**명령문에 관하여 앞에서 렬거한 기준들은 이 기준들로부터의 리탈에 관하여 결합한 어떤 기구와 함께 쏘프트웨어의 질을 개선할수 있으며 결국은 쏘프트웨어공학의 기본목적을 달성하게 된다.

１４．５．모듈의 재리용

재리용은 제8장에서 자세히 서술되였다. 사실 이 책에서는 재리용에 관한 자료들을 여러 곳에 서술하여 주었다. 왜냐하면 명세서, 계약서, 계획, 설계모듈의 부분들을 비롯하여 쏘프트웨어개발공정의 모든 단계들에서 나오는 요소들이 재리용되기때문이다. 이것이 바로 재리용에 관한 재료들을 어떤 특정한 단계에서가 아니라 이 책의 1편에 제시한 리유이다. 비록 모듈의 재리용이 모듈들이 재리용될수 있다는것보다 훨씬 일반적인 재리용형식이라고 할지라도 이 장에서 그것을 강조하기 위하여 모듈의 재리용에 대한 자료를 제시하지 않는다는것이 특별히 중요하다.

실현단계에서의 시험문제를 다음에 고찰한다.

１４．６．모듈시험실례선택

7장에서 설명한바와 같이 하나의 객체는 하나의 특정한 류형의 모듈이다. 그러므로 이 장에서는 이 책의 나머지부분에서와 마찬가지로 어떤 모듈에 대하여 성립하는것은 그 무엇이든지 일반적으로 객체들에도 적용한다. 그러나 일부 론의들은 객체들에 고유한것이다. 이 절과 다음절들에서는 객체들에 고유한 시험과 관련한 문제들이 일반적인 모듈시험의 범위내에서 강조하여 론의된다.

제6.6에서 지적한바와 같이 모듈들은 두가지 류형의 시험을 거치게 된다. 즉 모듈의 개발기간에 프로그람작성자에 의하여 진행되는 비형식적시험과 프로그람작성자가 모듈이 정확하게 기능하는것 같다고 만족한 이후에 SQA그룹에 의하여 진행되는 방법론적시험을 거친다. 이 방법론적시험에 대하여서는 아래에서 론의한다. 방법론적시험에는 또 두가지 기본류형이 있다. 하나는 비실행에 기초한 시험인데 여기서는 모듈이 어떤 개발팀에 의하여 검토된다. 다른 하나는 실행에 기초한 시험인데 여기서는 모듈이 시험실례에 대비하여 실행된다. 이러한 시험실례들을 선택하기 위한 기법들을 이제 고찰하기로 하자.

어떤 모듈을 시험하기 위한 최악의 방법은 우연시험자료를 리용하는것이다. 시험자는 건반앞에 앉아서 모듈이 입력을 요구할 때마다 임의의 자료를 입력한다. 앞으로 보게 되겠지만 가능한 모든 시험실례들중의 극히 일부분을 시험할 시간도 없는데 이러한 시험실례들은 쉽사리 10^{100}개이상에 도달할수 있다. 1,000개 되나마나한 실행할수 있는 시험실례들을 우연적으로 발생한 자료들에 랑비하기는 너무 아쉽다. 더 나쁜 경우로서 콤퓨터가 입력을 요구할 때 같은 자료에 대하여 한번이상 응답하는 경향도 있는데 이렇게 되면 훨씬 더 많은 시험실례들을 랑비하는것으로 될것이다. 그러므로 명백히 시험실례들은 체계적으로 구성하여야 한다.

１４．６．１．명세서시험과 코드시험

어떤 모듈을 시험하기 위한 시험자료들은 두가지 기본방식으로서 체계적으로 구성될수 있다. 첫번째 방법은 명세서에 대하여 시험하는것이다. 이 기법을 검은통(*black-box*)시험, 거동(*behavioral*)시험, 자료구동(*data-driven*)시험, 기능(*funtional*)시험, 입출력구동(*input/output-*

driven)시험라고도 부른다. 이 방법에서 코드자체는 무시된다. 시험실례를 작성할 때 리용하는 유일한 정보는 명세서이다. 다른 하나의 극단은 코드에 대한 시험인데 여기서는 시험실례들을 선택할 때 명세서를 무시한다. 이 방법을 유리통(*glass-box*)시험, 흰통(*white-box*)시험, 구조적(*structural*)시험, 론리구동(*logic-driven*)시험, 경로지향(*path-oriented*)시험이라고도 부른다(이렇게 많은 서로 다른 용어들이 있게 되는 원인을 알려면 다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

무엇때문에 같은 시험개념에 이토록 많은 서로 다른 이름이 주어 지는가를 묻는것은 응당하다. 쏘프트웨어공학에서 흔히 있는것처럼 같은 개념이 서로 다른 여러 연구자들에 의하여 독립적으로 발견되여 그 매 사람들이 자기 식의 용어를 창안하게 된다. 쏘프트웨어공학기업체가 이 단어들이 동일한 개념에 대한 서로 다른 이름이라는것을 깨달았을 때는 이미 시간이 늦게 된다. 즉 서로 다른 이름들이 쏘프트웨어공학용어집에 이미 들어 가 있게 되는것이다.

이 책에서는 용어 **검은통시험**(*black-box testing*)과 **유리통시험**(*glass-box testing*)을 리용하고 있다. 이 용어들은 특수하게 서술된다. 우리가 명세서를 시험할 때는 코드를 완전히 불투명한 하나의 검은통으로써 취급한다. 거꾸로 코드를 시험할 때에는 그 내부를 볼수 있어야 한다. 때문에 용어 **유리통시험**을 리용하였다. 우리는 용어 **흰통시험**(*white-box testing*)을 쓰지 않는다. 왜냐하면 이것이 약간 혼돈을 가져 오기때문이다. 즉 하나의 흰색칠을 한 통은 검은색칠을 한 통과 마찬가지로 불투명한것이다.

이제 명세서에 대한 시험으로부터 시작하여 이 두가지 기법들의 실행가능성에 대하여 고찰하자.

14.6.2. 명세서시험의 실현가능성

다음의 실례를 고찰하자. 어떤 자료처리제품에 대한 명세서가 5가지 류형의 대리권과 7가지 류형의 할인이 병합되여야 한다는것을 서술하고 있다고 하자. 대리권과 할인의 가능한 모든 결합을 시험하기 위하여서는 35개의 시험실례들이 필요하다. 대리권과 할인은 완전히 분리된 두개의 모듈로 계산되며 때문에 독립적으로 시험될수 있다는것을 말해야 소용이 없다. 즉 검은통시험을 진행할 때 제품은 하나의 검은통으로 취급되며 그러므로 그것의 내부구조는 전혀 상관이 없다.

이 실례는 각각 5개, 7개의 서로 다른 값을 취하는 두개의 인자 대리권과 할인만을 포함하고 있다. 임의의 현실적인 제품들은 수백가지 각이한 인자들을 가지게 된다. 가령 20개의 인자만이 있고 그 매 인자는 4개의 서로 다른 값만을 취할수 있다고 하여도 총 4^{20} 또는 1.1×10^{12}개의 서로 다른 시험실례들을 고찰하여야 한다.

1조개이상의 시험실례가 내재하고 있는 의미를 고찰하기 위하여 그것들을 모두 시험하는데 얼마나 오랜 시간이 걸리는가를 고찰하여 보자. 만일 어떤 프로그람작성자팀이

평균 30s당 1개의 속도로 시험실례들을 발생, 실행, 조사한다고 하면 제품을 빠짐없이 시험하는데 100만년이상 걸리게 될것이다.

그러므로 명세서를 빠짐없이 시험하는것은 조합폭발로 인하여 실천적으로 불가능하다. 즉 시험실례가 너무 많아서 고찰할수 없는것이다. 다음으로 코드에 대한 시험을 고찰하자.

14. 6. 3. 코드시험의 실현가능성

가장 일반적인 형식의 코드시험에서는 모듈을 통하는 매 경로가 적어도 한번 실행될것을 요구하고 있다. 이것이 실현불가능하다는것을 보기 위하여 그림 14-6에 보여 준 흐름도를 고찰하자. 이 흐름도가 아주 평범한것 같아도 거기에는 10^{12}개이상의 서로 다른 경로가 있다. 흐름도의 중심에 있는 6개의 그늘진 통그룹을 통과하는 가능한 경로는 5개이며 이로부터 흐름도를 통과하는 가능한 경로의 총 개수는 다음과 같다.

$$5^1+5^2+5^3+\cdots+5^{18} = 4.77 \times 10^{12} \qquad (14.1)$$

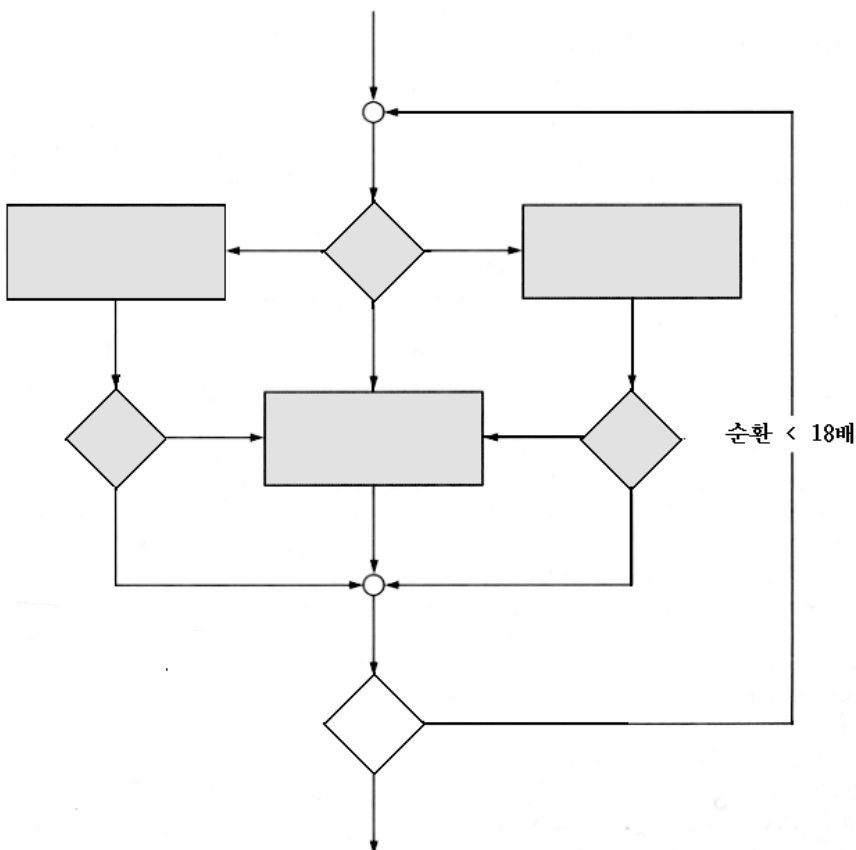



그림 14-6. 10^{12}개이상의 가능한 경로를 가진 흐름도

하나의 단독순환을 포함하고 있는 간단한 흐름도를 통하는 경로가 이렇게 많을수 있을진데 적당한 크기와 복잡도를 가지며 여러개의 순환을 가지는 어떤 모듈에서 서로 다른 경로의 총 개수가 얼마나 많겠는가를 어렵지 않게 상상할수 있다. 간단히 말하여 방대한 수량의 경로개수는 명세서에 대한 철저한 시험과 마찬가지로 코드에 대한 철저한 시험도 실현불가능하게 만든다.

코드시험이 문제시되는 추가적인 리유들도 있다. 코드에 대한 시험은 시험자가 매 경로를 시험해 볼것을 요구한다. 제품안의 모든 오유를 발견해 내지 않고 매 경로를 시험하는것이 가능할수도 있다. 즉 코드에 대한 시험은 믿을수 없다. 이것을 보기 위하여 그림 14-7에 보여 준 코드토막을 고찰하자[Myers, 1976]. 이 토막은 3개의 옹근수 x, y, z의 등가성을 시험하기 위하여 작성하였는데 여기서는 만일 3개 수의 평균값이 첫번째 수와 같다면 그 3개 수자는 등가이라고 하는 완전히 그릇된 가정을 리용하고 있다. 두개의 시험실례를 그림 14-7에 보여 준다. 첫번째 시험실례에서 3개 수의 평균값은 6/3 또는

```
if ((x+y+z)/3) ==x)
    print "x,y,z 는 값이 다 같다. ";
else
    print "x,y,z 는 다 같지 않다. ";
```

시험자료 1 : x = 1, y = 2, z = 3
시험자료 2 : x = y = z = 2

그림 14-7. 두 시험자료를 통하여 3개의 옹근수가 다 같은가를 결정하는 부정확한 코드부분

2로서 1과 같지 않다. 그러므로 프로그람은 시험자에게 x, y, z는 등가가 아니라고 정확하게 통보한다. 두번째 시험실례에서 옹근수 x, y, z는 모두 2이므로 계산된 평균값은 2로서 x의 값과 같으며 따라서 프로그람은 3개의 수가 등가이라고 정확하게 결론한다. 결국 이 프로그람을 통하는 두개의 경로는 오유가 발견되지 않고 시험되였다. 만일 x = 2, y = 1, z = 3과 같은 시험자료가 리용된다면 물론 오유가 드러나게 되였을것이다.

```
if ( d == 0)
    zeroDivisionRoutine ();
else
    x = n/d;
```

ㄱ)

```
x = n/d;
```

ㄴ)

그림 14-8. 나누기를 계산하는 두 코드부분

경로시험에서의 세번째 난점은 경로는 그 경로가 존재하여야만 시험될수 있다고 하는것이다. 그림 14-8의 ㄱ)에서 보여 준 코드토막을 고찰하자. 명백히 $d = 0$과 $d \neq 0$에 대응하는 두개의 경로가 시험되게 된다. 다음으로 그림 14-8의 ㄴ)의 단일명령문을 고찰하자. 여기에는 하나의 경로만이 있으며 이 경로는 오유가 발견되지 않고 시험될수 있다. 사실상 만일 프로그람작성자가 자기의 코드에서 $d = 0$의 시험을 생략하면 프로그람작성자는 모름지기 잠재적인 위험성을 알아 채지 못하며 $d = 0$의 경우는 프로그람작성자의 시험자료에 포함되지 않게 될것이다. 이 문제는 바로 이러한 류형의 오유를 발견할 임무를 지니고 있는 하나의 독립적인 쏘프트웨어품질보증그룹을 가지고 있어야 할 근거로 된다.

결론적으로 일부 자료들은 주어 진 경로를 시험하며 어떤 오유를 발견하고 또 다른 자료들은 같은 경로를 시험하며 오유를 발견하지 못하는것과 같은 제품이 존재하기때문에 이 실례들은 《제품의 모든 경로의 시험》을 믿을수 없다는것을 보여 주고 있다.

그러나 경로지향시험은 타당하다. 왜냐하면 경로지향시험이 오유를 드러낼수 있는 시험자료들을 선택하는것을 본성적으로 방해하지 않기때문이다.

조합폭발의 원인으로 하여 명세서를 빠짐없이 시험하거나 코드를 빠짐없이 시험하는것은 모두 실현불가능하다. 어떤 절충이 필요한데 그것은 될수록 많은 오유들을 두드러지게 하는 기법들을 리용하면서 한편 모든 오유들이 탐지되였다는것을 담보하는 방법은 없다는것을 인정하는것이다. 한가지 적당한 방법은 먼저 검은통시험(명세서시험)방법을 리용하고 그다음 유리통시험(코드시험)방법을 리용하여 추가적인 시험실례를 전개하는것이다.

14. 7. 검은통모듈시험기법

철저한 검은통시험방법은 일반적으로 수백만개의 시험실례들을 필요로 한다. 시험에서의 기교는 시험실례들을 처리할수 있는 작은 모임으로 분할하여 오유를 발견할 기회를 최소로 하는 한편 한개이상의 시험실례에 대하여 꼭 같은 오유가 발견되는것으로 인한 시험실례를 랑비할 기회를 최소로 하는것이다. 매 시험실례들은 사전에 발견되지 않은 어떤 오유를 발견할수 있도록 선택하여야 한다. 이와 같은 한가지 검은통기법은 등가성시험을 한계값분석과 결합하는것이다.

14. 7. 1. 등가성시험과 한계값분석

어떤 자료기지제품에 대한 명세서가 그 제품에 1부터 16,383($2^{14}-1$)사이의 임의의 개수의 기록을 조작할수 있어야 한다는것을 서술하고 있다고 하자. 만일 제품이 34개의 기록들과 14,870개의 기록들을 조작할수 있다면 그 제품이 이를테면 8,252개의 기록들을 애로없이 처리할 가망성은 충분하다. 사실 시험실례가 1부터 16,383개의 기록들사이에서 임의로 선택된다면 오유가 존재하는 경우 그 오유를 발견할 가능성도 마찬가지로 충분하다. 거꾸로 만일 제품이 1부터 16,383까지 범위에서 선택된 임의의 한개 시험실례에 대하여

정확하게 동작한다면 제품은 아마도 그 범위안의 임의의 다른 시험실례에 대해서도 동작할수 있을것이다. 1부터 16,383개까지의 기록범위는 하나의 등가클라스(*equivalence class*)를 이룬다. 즉 등가클라스는 그 클라스안의 임의의 한개 요소가 임의의 다른 요소와 마찬가지로 충분히 좋은 시험실례로 되게 하는 시험실례들의 모임이다. 보다 정확하게는 제품이 조작할수 있어야 하는 특정한 개수의 기록개수는 다음의 3가지 등가클라스를 정의한다.

등가클라스 1 : 1보다 적은 기록수
등가클라스 2 : 1부터 16,383개까지의 기록수
등가클라스 3 : 16,383개이상의 기록수

그러면 등가클라스기법을 리용한 자료기지제품의 시험은 매 등가클라스로부터 한개의 시험실례가 선택될것을 요구하게 된다. 등가클라스 2로부터 선택된 시험실례는 정확하게 처리되여야 하지만 반면에 클라스 1과 3으로부터 선택된 시험실례들에 대하여서는 오유통보가 인쇄되여야 한다.

성공한 하나의 시험실례는 앞에서 발견 못한 오유를 발견하게 된다. 이와 같은 오유를 찾아 낼 가능성을 최소화하기 위한 한가지 결정적인 방법은 경계값분석(*boundary value analysis*)이다. 경험은 어떤 등가클라스의 한쪽 경계면우에 놓이는 하나의 시험실례가 선택될 때에 오유를 발견할 가능성은 증가한다는것을 보여 주었다. 그렇기때문에 자료기지제품을 시험할 때 다음과 같은 7개의 시험실례들이 선택되여야 한다.

시험실례 1. 0개의 기록 : 등가클라스 1의 요소이며 경계값에 린접하여 있다.
시험실례 2. 1개의 기록 : 경계값
시험실례 3. 2개의 기록 : 경계값에 린접하여 있다.
시험실례 4. 723개의 기록 : 등가클라스 2의 요소
시험실례 5. 16,382개의 기록 : 경계값에 린접하여 있다.
시험실례 6. 16,383개의 기록 : 경계값
시험실례 7. 16,384개의 기록 : 등가클라스 3의 요소이며 경계값에 린접하여 있다.

이 실례를 입력명세서에 적용한다. 마찬가지로 강력한 한가지 기법은 출력명세서를 조사하는것이다. 실례로 2001년에 미국의 세금규약에 허용된 임의의 한 사람의 지불액으로부터의 최저사회보장면제액은 보다 정확하게는 최저늙은이, 생존자, 불구자보험(OASDI)면제액은 0.00딸라였고 최대면제액은 4,984.80딸라였는데 최대면제액은 총 국민소득 80,400딸라에 대응한것이다. 그러므로 어떤 지불명부제품을 시험할 때 지불액으로부터의 사회보장면제액에 관한 시험실례들은 정확하게 0.00딸라와 4,984.80딸라의 면제액을 생성할것으로 기대되는 입력자료를 포함하여야 한다. 이밖에 시험자료는 0.00딸라보다 작거나 4,984.80딸라보다 큰 면제액을 생성할수 있도록 설정되여야 한다.

일반적으로 입력명세서 또는 출력명세서에 렬거된 매 구역(R_1, R_2)에 대하여 다섯개의 시험실례들이 선택되여야 한다. 즉 이 시험실례들은 각각 R_1보다 작은 값, R_2과 같은

값, R_1보다는 크고 R_1보다는 작은 값, R_2와 같은 값 R_2보다 큰 값이 대응한다. 어떤 항목이 어떤 확정적인 모임의 원소이여야 한다면(실례로 입력은 하나의 문자여야 한다.) 두개의 등가클라스 즉 규정된 모임의 원소와 그 모임밖의 원소가 시험되여야 한다. 명세서가 어떤 정확한 값을 규정하고 있다면(실례로 응답뒤에 하나의 #기호가 뒤따른다.) 이때에도 두개의 등가클라스 즉 규정된 값과 그밖의 임의의 값이 존재하게 된다.

입력명세서와 출력명세서를 모두 시험하기 위하여 경계값분석과 함께 등가클라스들을 리용하는것은 발견하지 못한 여러가지 잠재적인 오유를 가지고 있는 상대적으로 작은 시험자료를 생성하는데서 매우 가치 있는 기법으로 되는데 이러한 잠재적인 오유들은 시험자료들을 선택하기 위하여 그리 강력하지 못한 기법들이 리용되는 경우에 여전히 숨겨져 있을수도 있게 된다.

등가성시험과정은 아래의 란에 종합한다.

등가성시험을 진행하는 방법

입력 및 출력명세서에 대하여

・매개의 구역(L, U)에 대하여 5개의 시험실례를 선택한다: 즉 L보다 작은것, L과 같은것, L보다 크고 U보다 작은것, U와 같은것, U보다 큰것을 선택한다.

・매 모임 S에 대하여 두개의 시험실례 즉 S의 요소와 S밖의 요소를 선택한다.

・매 정확한 값 P에 대하여 두개의 실례 즉 P와 그밖의 임의의것을 선택한다.

14．7．2．기능시험

한가지 다른 형식의 검은통시험방법은 모듈의 기능에 대한 시험자료에 의거하는것이다.

기능시험(*functional testing*)[Howden, 1987]에서 매개의 기능성항목 또는 모듈에서 실현된 기능은 동일시 된다. 콤퓨터화된 도매소제품을 위한 어떤 모듈에서 전형적인 기능들은《다음 자료기지기록을 취하시오.》 또는 《직접 취급할수 있는 주문량이 재주문량보다 적은가를 결정하시오.》이다. 어떤 무기조종체계에서 어떤 모듈은 기능 《전술방안을 계산하라.》를 포함할수 있다. 조작체계의 어떤 《파일이 비였는가를 결정하라.》가 하나의 기능으로 될수 있다.

모듈의 모든 기능들을 결정한후에 시험자료는 매 기능을 독립적으로 시험하기 위하여 분할된다. 이제 기능시험을 한걸음 더 전진시키자. 만일 모듈이 구조화된 프로그람작성에 대한 조종구조로 련결된 보다 낮은 준위의 기능들의 계층으로 구성된다면 기능시험은 재귀적으로 진행된다. 실례로 보다 높은 준위의 기능은 다음과 같은 형식

$$\begin{aligned}&< \textit{higher level function} >::= \textbf{if} < \textit{conditional expression} >\\&\qquad\qquad < \textit{lower-level function } 1> ;\\&\qquad \textbf{else}\\&\qquad\qquad < \textit{lower-level function } 2> ;\end{aligned} \quad (14.2)$$

으로 되여 있다면 <*conditional expression*>, <*lower-level function* 1>, <*lower-level function* 2>는 기능시험에 종속되기때문에 <*higher level function*>은 14.8.1에서 설명한 분기피복 즉 유리통기법을 리용하여 시험될수 있다. 이러한 형식의 구조적시험은 혼성식기법이다. 즉 저준위기능들은 검은통기법을 리용하여 시험되며 고준위기능들은 유리통기법을 리용하여 시험된다.

그러나 실천적으로 고준위기능들은 이와 같은 구조적인 형식으로 저준위기능들로부터 구성되지 않는다. 대신에 저준위기능들은 일반적으로 어떤 방식으로 얽혀 진다. 이러한 정황에서의 오유를 결정하기 위하여 약간 복잡한 절차인 기능분석(*functional analysis*)이 요구된다. 자세한 내용에 대하여서는 문헌 [Howden, 1987]을 보시오. 한가지 보다 복잡한 인자는 기능들이 특히 모듈의 경계와 일치하지 않는다는것이다. 그러므로 모듈시험과 통합시험사이의 구별이 명백치 않게 된다. 즉 하나의 모듈은 자기가 리용하는 다른 모듈의 기능을 동시에 시험하지 않고서는 시험될수 없다. 이러한 문제는 한 객체의 어떤 방법이 어떤 다른 객체의 한 방법에로 어떤 통보를 보낼 때(또는 그 방법을 호출할 때) 객체지향파라다임에서도 발생한다.

기능시험의 견지에서 모듈들사이의 우연적인 호상관계들은 관리자측에게 있어서 접수할수 없는 결과들을 가져 올수도 있다. 실례로 리정표와 최종기한이 약간 불명확하게 되여 쏘프트웨어프로젝트관리계획에 관하여 제품의 상태를 결정하기 어렵게 될수 있다.

14. 8. 유리통모듈시험기법

유리통기법들에서 시험실례들은 명세서가 아니라 코드의 조사에 기초하여 선택된다. 명령문피복, 분기피복, 경로피복을 비롯한 여러가지 서로 다른 류형의 유리통시험방법들이 있다.

14. 8. 1. 구조적시험: 명령문피복, 분기피복, 경로피복

가장 간단한 형식의 유리통시험방법은 명령문피복(*statement coverage*)이다. 즉 일련의 시험실례들이 실행되는 기간 매 명령문이 적어도 한번 실행되는 방법이다. 어느 명령문들이 계속 실행되고 있는가를 추적하기 위하여 CASE도구가 매 명령문이 시험렬에서 몇번 실행되였는가를 보여 주는 기록을 유지하게 된다. PureCoverage가 바로 이와 같은 도구의 한가지 실례로 된다.

이 방법의 약점은 모든 분기들의 결과가 정당하게 시험된다는 담보가 없다는것이다. 이것을 보여 주기 위하여 그림 14-9의 코드토막을 고찰하자. 프로그람작성자가 한개의 오유를 만들어 놓는다. 복합조건 s > 1 && t = = 0은 s > 1 || t = = 0으로 리해하여야 한다. 그림에 보여 준 시험자료는 명령문 x=9가 오유를 발생하지 않은채로 실행되게 한다.

명령문피복에 대한 한가지 개선된 방법이 분기피복(*branch coverage*)이다. 즉 일련의 시험들을 실행하여 모든 분기들이 적어도 한번 시험된다는것을 확인하는 방법이다. 여기서도 역시 시험자가 어느 분기들이 시험되였는가 또는 시험되지 않았는가를 추적할수 있

도록 하는 어떤 도구가 일반적으로 요구된다. Generic Coverage Tool(*gct*)은 C프로그람작성을 위한 분기피복의 한가지 실례이다. 이와 같은 명령문 또는 분기피복기법을 구조적 시험(*structural tests*)이라고 부른다.

```
if ( s > 1 && t == 0)
    x = 9;

시험자료 ; s = 2,  t = 0.
```

그림 14-9. 시험자료를 가진 코드부분

가장 강력한 형태의 구조적시험은 경로피복(*path coverage*)이다. 즉 모든 경로들을 시험하는것이다. 앞에서 보여 준바와 같이 어떤 순환을 가진 쏘프트웨어제품에서 경로의 수는 사실상 매우 클수 있다. 이로부터 연구자들은 분기피복기법을 리용할 때보다 더 많은 오유들을 발견하면서 시험될 경로의 수를 줄이는 방법을 연구하여 왔다. 경로를 선택하기 위한 한가지 선택기준은 시험실례를 선형코드순서렬로 제한하는것이다[Woodward, Hedley, and Hennell, 1980]. 이를 위하여 먼저 조종흐름이 비약할수 있는 점들의 모임 L을 찾아 낸다. 모임 L은 입력 및 탈퇴점들과 **if** 또는 **goto**와 같은 분기명령문들을 포함한다. 그러면 선형코드순서렬은 L의 요소에서 시작하고 L의 요소에서 끝나는 경로들로 된다. 이 방법은 모든 경로들을 시험하지 않고 많은 오유들을 발견한다는 점에서 성공적이였다.

시험될 경로의 수를 줄이기 위한 또 한가지 방법은 정의와 리용사이의 모든 경로피복(*all-definition-use-path-coverage*)이다[Rapps and Weyuker, 1985]. 이 기법에서는 원천코드안에서 실례를 들어 어떤 변수 pqr의 출현은 pqr = 1 또는 read (pqr)와 같은 변수의 정의(*definition*)로 표식되든가 y = pqr＋3 또는 **if** (pqr < 9) error B()와 같은 변수의 리용(*use*)으로써 표식된다. 어떤 변수의 정의와 그 정의의 리용사이의 모든 경로들은 현재 하나의 자동도구에 의하여 식별된다. 결국 하나의 시험실례가 이와 같은 매개 경로에 설정된다. 정의와 리용사이의 모든 경로피복은 상대적으로 적은 시험실례들로서 많은 오유들을 발견하는 한가지 매우 우월한 시험기법으로 된다. 그러나 이 방법은 경로개수의 웃 한계가 2^d로 된다는 부족점을 가지고 있다. 여기서 d는 제품에서의 결정명령문(분기)의 개수이다. 웃한계를 보여 주기 위한 실례를 구성할수 있다. 그러나 인공적인 실례와는 반대되게 실지제품들에서는 이러한 웃한계가 실현되지 않으며 실제적인 경로의 개수는 d에 비례한다는것이 밝혀 졌다[Weyuker, 1988a].

달리 말하면 이 방법에 요구되는 시험실례의 개수는 리론적인 웃한계보다 훨씬 더 작다. 결국 정의와 리용사이의 모든 경로피복은 실천적인 시험실례선택기법으로 된다.

구조적시험를 리용할 때 시험자는 간단히 어떤 특정한 명령문, 분기 또는 경로를 사용하게 될 시험실례를 들고 나오지 않을수도 있다. 이렇게 되면 모듈안에 어떤 실현불가능한 경로(《죽은 코드》) 즉 임의의 입력자료에 대하여 실행될수 없는 경로가 존재하게 된다. 그림 14-10은 실현불가능한 경로의 두가지 실례를 보여 준다. 그림 14-10의 ㄱ)에서 프로그람작성자는 하나의 미누스기호를 생략하였다. 만일 k가 2보다 작다면 k는 3보다 클

수 없으며 그러므로 명령문 x = x×k는 실현될수 없다. 류사하게 그림 14-10의 ㄴ)에서 j는 0보다 작을수 없으므로 명령문 total＝total＋value[j]는 결코 실현될수 없다. 즉 프로그람작성자는 j<10을 시험할 작정이였지만 건반입력오유가 만들어 졌다. 명령문피복을 리용하는 시험자는 명령문이 실현될수 없다든가 오유들이 발견될수 없다는것을 인차 깨달을것이다.

```
if ( k < 2)
{
      if ( k > 3)                [ k > −3 이여야 한다]
                 ↑
            x = x * k;
}
```

ㄱ)

```
for ( j = 0; j<0; j ++)          [ j < 10 이여야 한다]
                ↑
      total = total + value[j];
```

ㄴ)

그림 14-10. 실행불가능한 경로의 두 실례

14. 8. 2. 복잡도척도

품질보증관점은 유리통시험에 대한 또 한가지 기법을 제공한다. 어떤 관리자에게 모듈 m1이 모듈 m2보다 더 복잡하다고 알려 졌다고 하자. 용어 《복잡하다.》가 정의되는 정확한 방법에 관계없이 관리자는 직관적으로 m1은 m2보다 더 많은 오유를 가질수 있다고 믿을것이다. 이러한 착상으로부터 콤퓨터과학자들은 어느 모듈이 가장 오유를 포함할 가능성이 있는가를 결정하기 위한 한가지 수단으로서 쏘프트웨어의 복잡도에 관한 여러가지 척도들을 개발하였다. 만일 어떤 모듈의 복잡도가 터무니없이 높다는것이 알려 지게 되면 관리자는 틀리기 쉬운 모듈을 수정하기 보다는 처음부터 시작하는것이 값도 눅고 더 빠를수 있다는 리유로 하여 그 모듈을 재설계하고 재실현할것을 지시할수 있다.

오유의 개수를 예보하기 위한 한가지 단순한 척도는 코드의 행이다. 이때 기초로 하는 가정은 코드의 한개 행이 하나의 오유를 포함할 일정한 확률 p가 존재한다는것이다. 만일 시험자가 코드의 한 행이 평균 2%의 오유를 포함할 가능성을 가진다고 믿고 있으며 시험하는 모듈의 길이가 100행이라면 그 모듈은 2개의 오유를 포함할것이며 이 두개의 길이를 가지는 모듈은 4개의 오유를 포함할수 있다고 추측된다는것을 의미한다. 다까하시(Takahashi)와 가마야스(Kamayachi)는 물론 바실리(Basili)와 후첸스(Hutchens)는 오유의 개수가 실지로 전체 제품의 크기에 관계된다는것을 보여 주었다[Basili and Hutchens, 1983; Takahashi and Kamayachi, 1985].

제품의 복잡도에 대한 척도에 기초하여 보다 정교한 오유예보기를 찾아 내려는 시도들이 있었다. 한가지 전형적인 경쟁자는 2분결정(술어)의 개수에 1을 더한 맥케브(McCabe)의 주기적복잡도이다[McCabe, 1976]. 13.12에서 서술한바와 같이 주기적복잡도는 본질에 있어서 모듈에서 분기의 개수이다. 따라서 주기적복잡도는 어떤 모듈의 분기피복에 요구되는 시험실례들에 대한 척도로서 리용될수 있다. 이것이 이른바 구조적시험의 기초이다[Watson and McCabe, 1996].

맥케브의 척도는 코드의 행처럼 그렇게 쉽게 계산될수 있다. 일부 실례들에서 이것이 오유를 예보하기 위한 한가지 좋은 척도로 된다는것이 제시되였다. 즉 *M*의 값이 크면 클수록 어떤 모듈이 오유를 포함한 가능성은 더 크다. 실례로 월쉬는 함선전투체계인 Aegis체계안에 있는 276개 모듈들을 분석하였다[Walsh, 1979]. 그는 주기적복잡도 *M*을 측정하고 *M*이 10보다 크거나 같은 23%의 모듈들이 53%의 오유를 발견하였다는것을 밝히였다. 이밖에 *M*이 10보다 크거나 같은 모듈들이 그보다 작은 *M*값을 가진 모듈들보다 코드행당 21% 더 많은 오유를 포함하고 있었다. 그러나 맥케브척도의 타당성은 그 리론적근거와 문헌 [Shepperd, 1988b, and. Sheppered and Ince, 1994]에서 렬거된 여러가지 서로 다른 실험들에 기초하여 심히 문제시되였다.

할스테드(Halstead)의 쏘프트웨어과학척도들[Halstead, 1977]도 오유예보를 위하여 리용되였다. 쏘프트웨어과학에서 4가지 기본요소들중의 두가지는 모듈에서 개별적연산자들의 수 *n*1과 개별적연산수의 개수 *n*2이다. 전형적인 연산자들에는 +, ×, **if**, **goto** 등이 포함되며 연산수는 사용자정의변수 또는 상수들이다. 다른 두가지 기본요소는 연산자의 총수 *N*1과 연산수의 총 수 *N*2이다. 그림 14-10의 ㄱ)의 코드토막을 그림 14-11에서 다시 인용하였다. 그림 14-11로부터 코드토막이 10개의 개별적인 연산자들과 4개의 개별적연산수를 가지고 있다는것을 알수 있다. 연산자와 연산수의 총 개수는 각각 13과 17이다. 그러면 이 4개의 기본요소들은 오유예보척도를 위한 입력으로서 리용된다[Ottenstein, 1979].

```
if ( k < 2)
{
    if (k > 3)
        x = x * k;
}
```

명백한 연산자;
if (<) { > = * ; }

명백한 연산수;
K 2 3 x

명백한 연산수 n1 의 수 = 10
명백한 연산수 n2 의 수 = 4
연산자 *N*1 의 총수 = 13
연산자 *N*2 의 총수 = 7

그림 14-11. 그림14-10의 ㄱ)의 코드부분에 적응된 쏘프트웨어과학척도

무싸(Musa), 이아니노(Iannino), 오꾸모또(Okumoto)는 오유밀도에 적용할수 있는 자료를 분석하였다[Musa, Iannino, Okumoto, 1987]. 그들은 할스테드와 맥케브를 비롯한 다수의 복잡성척도들이 코드행의 개수 더 정확하게는 전달가능하고 실행가능한 원천명령들의 개수와 크게 상관되여 있다는것을 결론하였다. 달리 말하면 연구자들이 어떤 모듈 또는 제품의 복잡도라고 자기들이 믿고 있는것을 측정할 때 얻게 되는 결론은 대체로 코드행의 개수에 대한 어떤 반영 즉 오유의 개수와 강하게 상관되여 있는 어떤 측도일수 있다는 복잡도와 관련된 다른 문제가 문헌 [Shepperd and Ince, 1994]에서 론의된다. 더우기 할스테드의 척도를 Java 또는 C++와 같은 현대언어에 적용하는 경우의 문제들도 있다. 실례를 들면 어떤 구축자가 연산자인가 또는 연산수인가 하는것이 명백하지 않다.

14. 9. 코드관통심사회의와 검토

6.2에서는 관통심사회의와 검토에 관한 하나의 유력한 실례를 제시하였다. 그와 마찬가지의 론거가 코드관통심사회의와 검토에 관하여 성립한다. 간단히 말하면 이 두가지 비실행에 기초한 기법들의 오유발견능력은 신속하고 철저하며 신속한 오유발견을 할수 있게 한다. 코드관통심사회의 또는 검토에 요구되는 추가적인 시간은 통합단계에서 보다 적은 오유로 인하여 얻게 되는 생산성의 증대로서 충분히 보상된다. 더우기 코드검토사는 교정유지정비비용을 95%까지 감소시키였다[Crossman, 1982].

코드검토를 진행해야 할 다른 한가지 리유는 다른 실행에 기초한 시험(시험실례)이 다음의 두가지 측면에서 극히 비경제적이기때문이다. 첫째로, 시간을 랑비하는것이다. 둘째로, 검토는 실행에 기초한 시험보다 앞선 생명주기단계들에서 오유를 발견하고 수정할수 있게 한다. 그림 1-5에 반영되여 있는것처럼 오유를 보다 빨리 발견하고 수정하면 할수록 비용이 더 적게 든다. 시험실례들의 실행에 많은 비용이 든 한가지 극단한 실례는 NASA의 아폴로계획(*Appollo Program*)인데 여기에서는 쏘프트웨어를 위한 비용의 80%가 시험에 소비되였다[Dunn, 1984].

관통심사회의와 검토의 진행에 관한 그이상의 론거들은 다음절에서 론의한다.

14. 10. 모듈시험기법들의 비교

많은 연구들에서 모듈시험을 위한 방략들을 비교하였다. 마이어스(Myers)는 검은통시험, 검은통과 유리통시험의 결합, 3인코드관통심사회의를 비교하였다[Myers, 1978a]. 이 실험은 많은 경험을 가진 59명의 프로그람작성자들이 동일한 제품을 시험하는것으로써 진행되였다. 이 세가지 기법들이 모두 오유발견에서는 동등한 결과를 가지였지만 코드관통심사회의가 다른 두 방법보다 비용도 적고 효과적이라는것이 증명되였다. 황(Hwang)은 검은통시험, 유리통시험, 사람에 의한 코드조사를 비교하였다[Hwang, 1981]. 이 세가지 기법이 모두 동등한 효과성을 가진다는것이 밝혀 졌으며 매 기법들은 자기의 우점과 약점을 가지고 있다.

한가지 중요한 실험이 와씰리, 쎌비의 지도밑에 진행되였다[Basili and Selby, 1987]. 이 방법에서는 황의 실험과 마찬가지로 검은통시험, 유리통시험, 1인코드조사가 비교되였

다. 담당자는 32명의 프로그람전문가와 42명의 우수한 대학생들이였다. 매 사람이 매 시험기법을 한번씩 리용하여 3개의 제품을 시험하였다. 분수차례곱설계[Basils and Weiss, 1984]가 그 제품들이 각이한 실험참가자들에 의하여 시험된 각이한 방법을 보상하기 위하여 리용되였다. 즉 참가자들은 같은 제품을 한가지이상의 방법으로 시험하지 않았다. 두 그룹으로부터 각이한 결과들이 얻어 졌다. 전문프로그람작성자들은 다른 두가지 기법보다도 코드읽기를 할 때 더 많은 오유를 발견하였으며 오유발견속도도 더 빨랐다. 두개의 우수한 대학생그룹이 이 실험에 참가하였다. 첫번째 대학생그룹은 3가지 기법에서 본질적인 차이가 없었으며 두번째 그룹에서는 코드조사와 검은통시험이 동등한 효과를 가지였으며 이 두 기법이 모두 유리통시험보다 우월하였다.

그러나 대학생들이 오유를 발견하는 속도는 이 세가지 기법에서 모두 같았다. 총적으로 코드읽기는 다른 두 기법보다 더 많은 대면부적오유들을 발견하였으며 반면에 검은통시험은 조종오유들을 찾아 낼 때 가장 성공적이였다. 이 실험으로부터 이끌어 낼수 있는 중요한 결론은 코드검토는 적어도 오유를 발견할 때에 유리통시험 및 검은통시험과 마찬가지로 성공적이라는것이다.

이 결론을 훌륭히 리용하는 한가지 개선된 기법이 무진실쏘프트웨어개발기법이다.

14. 11. 무진실

무진실기법(*Cleanroom technique*)[Linger 1994]은 증식생명주기모형(3.4), 명세서와 설계에 대한 형식적기법들, 코드읽기[Mills, Dyer, and Linger, 1987]와 코드관통심사회의와 검토(14.9)와 같은 비실행에 기초한 모듈시험기법들을 비롯하여 여러가지 각이한 쏘프트웨어개발수법들을 결합한것이다. 무진실기법의 중요한 측면은 어떤 모듈은 그것이 어떤 검토를 거쳤을 때에야 비로서 콤파일된다는것이다. 즉 모듈은 비실행에 기초한 시험이 성공한후에만 콤파일 되여야 한다.

이 기법은 여러가지 큰 성공을 가져 왔다. 실례로 미해군수중체계쎈터(U.S.Naval Underwater Systems Center)에서는 무진실기법을 리용하여 원형자동문서화체계를 개발하였다[Trammel, Binder, and Snyder, 1992].

설계가 《기능검증》을 거치는 과정에 즉 정확성증명기법들이 리용된 검토과정에 모두 18개의 오유가 발견되였다(6.5). 6.5.1에서 제시된 검토와 같은 비형식적증명들이 될수록 많이 리용되였다. 한편 완전한 수학적증명들은 검토참가자들이 검토되고 있는 설계부분들의 정확성을 믿지 못하는 경우에만 전개되였다. 1,820행의 FoxBASE코드에 대한 관통심사회의기간에 다른 19개의 오유들이 발견되였다. 코드가 콤파일 되였을 때 그 어떤 콤파일오유도 없었다. 더우기 실행기간에는 전혀 오유가 없었다. 이것은 비실행에 기초한 시험기법의 능력을 더욱 강조하여 주고 있다.

이것은 확실히 하나의 인상적인 결과이다. 그러나 지적된바와 같이 소규모쏘프트웨어제품들에 적용한 결과들은 대규모쏘프트웨어제품들에로 반드시 확대되는것은 아니다. 그러나 무진실기법의 경우에 보다 규모가 큰 제품들에 대한 결과도 역시 인상적이다. 이와 관련한 척도는 오유시험률(*testing fault rate*) 즉 KLOC(1,000개의 코드행)당 발견된 오

유의 총 개수인데 이것은 쏘프트웨어산업에서 상대적으로 일반적인 한가지 척도이다. 그러나 무진실기법이 전통적인 개발기법들에 대립하여 리용될 때 이 척도가 계산되는 방식에서 한가지 중요한 차이가 존재한다.

5.6에서 지적한 바와 같이 전통적인 개발기법들이 리용될 때 이런 모듈은 그것이 개발될 때에 그 작성자에 의하여 형식적으로 시험되고 그 다음에 SQA그룹에 의하여 방법론적으로 시험된다. 코드를 개발하는 동안 프로그람작성자에 의하여 발견된 오유들은 기록되지 않는다. 그러나 모듈이 프로그람작성자개인의 워크스테이션을 떠나서 실행 및 비실행에 기초한 시험를 위하여 SQA그룹에 넘겨 진 때부터 계수기에 발견된 오유의 개수를 보관해 둔다. 이와 반대로 무진실이 리용될 때에는 《시험오유》들은 콤파일을 할 때부터 계수된다. 그다음 오유계산은 실행에 기초한 시험전기간에 계속된다. 달리 말하면 전통적개발기법들이 리용될 때에는 프로그람작성자에 의하여 비형식적으로 발견된 오유들은 시험오유률의 계산에 들어 가지 않는다. 무진실기법이 리용될 때에는 검토기간과 콤파일이전의 기타 비실행에 기초한 시험절차기간에 발견된 오유들은 기록되지만 그것들은 시험오유률의 계산에 들어 가지 않는다.

무진실을 리용한 17개의 제품에 대한 보고가 문헌 [Linger, 1994]에 제시되였다. 실례로 무진실은 350,000행의 Ericsson Telecom OS32조작체계를 개발하는데 리용되였다. 이 제품은 70개의 팀이 참가하여 18개월동안에 개발하였다. 시험오유률은 KLOC당 1.0오유에 불과하였다. 또 다른 제품은 앞에서 설명한 신속원형자동문서화체계이다. 그것의 시험오유률은 1,820행의 프로그람에 대하여 KLOC당 0.0오유이다. 17개 제품들을 모두 합치면 거의 1백만행에 이른다. 무게붙은 평균오유시험률은 KLOC당 2.3오유인데 링게르(Linger)는 이것을 상당한 품질달성으로서 서술하고 있다. 이 자랑은 명백히 과장된것이 아니다.

14.12. 객체시험에서 제기될수 있는 문제

객체지향파라다임을 리용할것을 주장하는 많은 리유들중의 하나는 그것이 시험에 대한 요구를 줄인다는것이다. 계승을 통한 재리용은 객체지향파라다임의 또 하나의 우점이다. 어떤 클라스가 일단 시험되면 그것은 재시험할 필요가 없다는 증거가 성립한다. 더우기 이와 같은 어떤 시험된 클라스의 부분클라스내에서 정의된 새 방법들은 시험되여야 하지만 계승된 방법들은 더이상 시험할 필요가 없다.

사실 이 두가지 주장은 부분적으로만 성립하였다. 이밖에 객체들의 시험은 객체지향에 고유한 일정한 문제들도 발생시켰다. 여기에서는 이러한 문제점들을 론의한다.

먼저 클라스들과 객체들의 시험과 관련한 한가지 문제점을 명백히 할 필요가 있다. 7.7에서 설명한바와 같이 하나의 클라스는 계승을 지원하고 있는 하나의 추상자료류형이며 하나의 객체는 어떤 클라스의 하나의 실례이다. 즉 하나의 클라스는 구체적인 실체를 가지지 않으며 반면에 하나의 객체는 어떤 특정한 환경내에서 실행되는 하나의 물리적인 코드토막이다. 그러므로 어떤 클라스에 실행에 기초한 시험을 진행하는것은 불가능하며 다만 어떤 검토와 같은 비실행에 기초한 시험만을 진행할수 있다.

정보은페와 대다수방법들이 상대적으로 적은 행수의 코드로 이루어 진다는 사실은

시험에 한가지 중요한 영향을 줄수 있다. 먼저 구조화파라다임을 리용하여 개발된 어떤 제품을 고찰하자. 현재 이와 같은 제품은 일반적으로 약 50개의 실행가능한 명령들로 구성된다. 하나의 모듈과 제품의 나머지 모듈사이의 대면부는 인수목록이다. 인수들는 두가지 종류인데 하나는 모듈이 호출될 때 그 모듈에 공급되는 입력인수들이고 다른 하나는 모듈이 호출하는 모듈에 조종을 넘길 때 그 모듈이 귀환하는 출력인수들이다. 어떤 모듈을 시험하는것은 입력인수들에 값들을 주고 그 모듈을 호출하며 그다음 출력모듈의 값을 예상하는 시험결과와 비교하는 과정으로 이루어 진다.

반대로 어떤 《전형적인》객체는 약 30개의 방법들을 포함하고 있는데 그 대부분은 상대적으로 규모가 작으며 흔히 2～3개의 실행가능한 명령들이다[Wilde, Matthews and Huitt, 1993]. 이 방법들은 호출자에게 값을 귀환하지 않지만 오히려 그 객체의 상태를 변화시킨다. 즉 이러한 방법들은 그 객체의 속성(상태변수)들을 변경시킨다. 여기서 난점은 상태의 변경이 정확하게 수행되였다는것을 시험하기 위하여서는 그 객체에 추가적인 통보를 보내는것이 필요하다는것이다. 실례로 1.6에서 고찰한 은행체계객체를 고찰하자. 방법 deposit의 의미는 상태변수 account balance의 값을 증가시키는것이다. 그러나 정보은페로 인하여 어떤 특정한 deposit작용이 정확하게 실행되였는가를 시험하기 위한 유일한 방도는 방법 deposit를 호출하기전과 호출한후에 모두 방법 determine balance를 호출하여 은행결산이 어떻게 변화되였는가를 보는것이다.

만일 그 객체가 모든 상태변수들의 값을 결정하기 위하여 호출될수 있는 방법들을 포함하지 않고 있다면 정황은 더욱 불리하다. 한가지 대안은 이 목적을 위하여 추가적인 방법을 포함시키고 그다음 추가적인 콤파일을 리용하여 그것들을 시험목적외에는 리용할 수 없다는것을 확인하는것이다(C++에서 이것은 **#ifdef**를 리용하여 실현된다.). 시험계획(9.6)은 매 상태변수의 값을 시험기간에 호출가능하여야 한다는것을 규정할것이다. 이 요구를 만족시키기 위하여 상태변수의 값들을 귀환하는 추가적인 방법들이 설계단계에서 련관된 클라스들에 추가되여야 할수도 있다. 그 결과로 작용가능한 상태변수들의 값을 질문하는것으로써 어떤 객체의 어떤 특정한 방법을 호출하는 결과를 시험하는것이 가능하게 될것이다.

매우 놀랍게도 계승된 방법은 여전히 시험되여야 한다. 즉 어떤 방법이 적당하게 시험되였다고 할지라도 그 방법은 어떤 부분클라스에 의하여 계승되고 변경되지 않을 때 철저한 시험을 진행할것을 요구할수도 있다. 후자의 문제점을 보기 위하여 그림 14-12에 보여 준 클라스계층들을 고찰하자.

기초클라스 **RootedTree**안에서 두개의 방법 displayNodeContents와 printRoutine이 정의되였는데 방법 displayNodeContents는 방법 printRoutine을 리용하고 있다.

다음으로 부분클라스 **BinaryTree**를 고찰하자. 이 부분클라스는 그것의 기초클라스 **RootedTree**로부터 방법 printRoutine을 계승하고 있다. 이밖에 하나의 새로운 방법 displayNodeContents이 정의되는데 이것은 **RootedTree**에서 정의된 방법을 무효로 한다. 이 새로운 방법은 여전히 PrintRoutine을 리용하고 있다. Java표식법에서는 BinaryTree. displayNodeContents가 RootedTree.printRoutine을 리용하고 있다.

```
class  RootedTree
{
    …
  void  displayNodeContents  (Node  a);
  void  printRoutine  (Node  b);
 //
 //   방법  displayNodeContents는   방법  printRoutine을  리용한다
 //
 …
}

class  BinaryTree  extends  RootedTree
{
    …
   void  displayNodeContents  (Node  a);
 //
 //   이  클라스에서  정의된  방법  displayNodeContents는
 //   클라스  RootedTree로  부터  계승된  방법  printRoutine  을  리용한다
 //
    …
}

class  BalancedBinaryTree  extends  BinaryTree
{
      …
   void  printRoutine  (Node  b);
 //
 //   (BinaryTree  로  부터  계승된)방법  displayNodeContents는
 //   클라스  BalancedBinaryTree안에  있는  방법  printRoutine의  이러한
 //   국부적인  판본을  리용한다.
 //
       …
}
```

그림 14-12. 나무계층의 Java실현

다음으로 부분클라스 **BalancedBinaryTree**를 고찰하자. 이 부분클라스는 그 상위클라스 **BinaryTree**로부터 방법 displayNodeContents를 계승하고 있다. 그러나 RootedTree에서 정의된것을 무효로 하는 하나의 새로운 방법 PrintRoutine이 정의된다. displayNodeContents가 **BalancedBinaryTree**의 문맥범위내에서 printRoutine을 리용할 때 C++와 Java의 령역규칙들은 printRoutine의 국부판본이 리용되게 된다는것을 규정하고 있다. Java표식법에서 방법 BinaryTree.displayNodeContents가 **BalancedBinaryTree**의 사전령역내에서 호출될 때 그것은 방법 BalancedBinaryTree.printRoutine을 리용한다. 그러므로 displayNodeContents가 **BinaryTree**의 실례범위내에서 호출될 때 실행된 실제코드(방법 printRoutine)는 displayNode Contents가 **BalancedBinaryTree**의 실례범위내에서 호출될 때 실행된 실제코드와 다르다.

이것은 방법 displayNodeContents 그자체가 **BinaryTree**로부터 **BalancedBinaryTree**에 의하여 계승되고 변경되지 않음에도 불구하고 성립한다. 그러므로 방법 displayNodeContents가 하나의 **BinaryTree**객체내에서 철저히 시험되였다고 할지라도 그 방법은 어떤 **BalancedBinaryTree**환경내에서 재리용될 때 처음부터 재시험되여야 한다. 문제가 좀더 복잡하게 되여 방법을 각이한 시험실례로써 재시험하는것이 왜 필요한가 하는 리론적근거가 있다[Perry and Kaiser, 1990].

이러한 복잡화가 객체지향파라다임을 포기할 리유로는 되지 않는다. 첫째로, 복잡화는 다만 방법(실례에서 displayNodeContents와 printRoutine)들의 호상작용을 통하여 생겨난다. 둘째로, 이러한 재시험이 언제 요구되는가를 결정하는것이 가능하다[Harrold, McGregor, and Fitzpatrick, 1992]. 이런 클라스의 한 실례가 철저히 시험되였다고 하자. 그러면 어떤 부분클라스의 임의의 새로운 방법 또는 재정의된 방법들은 그것들이 다른 방법들과 호상작용하는것으로 인하여 재시험되도록 기발표식을 한 방법들과 함께 시험될 필요가 있다. 간단히 말하면 객체지향파라다임의 리용이 시험에 대한 요구를 크게 줄인다는 주장은 타당하다.

다음으로 모듈시험에 대한 몇가지 관리적의미에 대하여 론의한다.

14. 13. 모듈시험의 관리측면

매 모듈의 개발기간에 내려야 할 하나의 중요한 결정은 그 모듈을 시험하는데 얼마만한 시간을 소비하여야 하는가 또 그로 인하여 얼마만한 자금을 소비하여야 하는가 하는것이다. 쏘프트웨어공학에서 그렇게 많이 론의되고 있는 경제적문제들과 마찬가지로 비용 대 리득분석(5.2)이 유용한 역할을 놀수 있다. 실례로 정확성증명에 드는 비용이 어떤 특정한 제품이 명세서를 만족시킨다는 담보에 인한 비용을 초과하는가에 관한 결정은 비용 대 리득분석에 기초하여 채택될수 있다. 비용 대 리득분석은 또한 부적당한 시험으로 하여 생성된 정식제품에서의 오유수정비용 대 추가적인 시험실례의 실행비용을 비교하는데 리용될수 있다.

어떤 특정한 모듈의 시험을 계속하겠는가 또는 모든 오유들이 사실상 제거되였는가를 결정하기 위한 또 한가지 방법이 있다. 믿음성분석기법들이 얼마만한 오유들이 남아있는가에 대한 통계적추정을 제공하는데 리용될수 있다. 남아 있는 오유개수에 대한 통계적추정량을 결정하기 위하여 여러가지 각이한 기법들이 제안되였다. 이 기법의 기초를 이루고 있는 기본착상은 다음과 같다. 어떤 모듈이 1주일동안 시험된다고 가정하자. 월요일에 23개의 오유가 발견되고 화요일에 7개가 발견된다. 수요일에는 오유가 5개 더 발견되며 목요일에는 2개 발견되고 금요일에는 오유가 발견되지 않는다. 오유발견률이 하루 23개로부터 0에로 안정하게 감소되기때문에 대부분의 오유가 발견된것 같으며 따라서 그 모듈의 시험을 중지하게 된다. 코드에 더이상 오유가 없을 확률을 결정하기 위하여서는 이 책의 독자들에게 필요한것 이상의 높은 준위의 통계수학이 요구된다. 그러므로 여기서는 더 자세히 론의하지 않는다. 믿음성분석에 흥미를 가지는 독자들은 문헌 [Gray, 1992]를 참고할수 있다.

１４．１４．모듈을 오유수정하지 않고 재작성하는 경우

SQA그룹의 한 성원이 어떤 오유(오유출력)를 발견하였을 때 앞에서 서술한바와 같이 그 모듈은 오유수정 즉 오유의 시험과 코드의 정정을 위하여 원래의 프로그람작성자에게 귀환되여야 한다. 어떤 경우에는 그 모듈을 버리고 원래의 프로그람작성자 또는 개발팀의 수준이 보다 높은 다른 프로그람작성자가 모듈을 처음부터 다시 설계하고 다시 코드작성하는것이 오히려 더 나을수 있다. 왜 이것이 필요한가를 보기 위하여 그림 14-13을 고찰하자. 그림에서 그라프는 어떤 모듈에서 그이상의 오유가 존재할 확률은 그 모듈에서 이미 발견된 오유의 개수에 비례한다는 비직관적인 개념을 보여 주고 있다[Myers, 1979]. 그 리유는 무엇때문인가를 보기 위하여 두개의 모듈 m1과 m2를 고찰하자. 두 모듈은 근사적으로 같은 길이를 가지며 둘 다 같은 시간동안 시험되였다고 가정하자. 그리고 m1에서는 다만 2개의 오유가 발견되였지만 m2에서는 48개의 오유가 발견되였다고 가정하자. m2에서는 m1보다 찾아 내야 할 오유가 더많이 남아 있는것 같다. 더우기 m2에 대한 추가적인 시험과 오유수정은 오랜 시간이 걸리는 공정으로 될것 같으며 m2이 여전히 완전하지 못하다는 의심이 남아 있게 될것이다. 단기실행과 장기실행에서 모두 m2를 버리고 그것을 다시 설계하고 다시 코드작성하는것이 오히려 낫다.

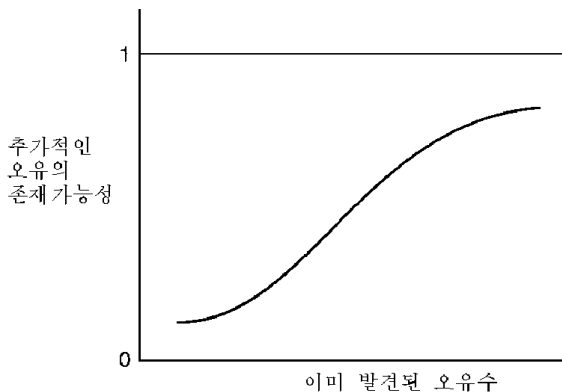



그림 14-13. 오유가 발견될 확률이 이미 발견된 오유수에
비례한다는것을 보여 주는 그라프

모듈에서 오유의 분포는 확실히 균등하지 않다. 마이어스는 OS/370에서 사용자에 의하여 찾아 진 오유들의 실례를 보여 주었다[Myers, 1979]. 여기에서는 47%의 오유들이 다만 4%의 모듈들과만 관련되여 있다는것이 밝혀 졌다. 엔드레스(Endres)가 IBM 연구소에서 DOS/VS(Release28)의 내부시험과 관련하여 진행한 보다 앞선 연구와 도이췰란드의 뵐링겐(Böblingen)의 연구는 우와 류사한 오유분포에서의 불균등성을 보여 주었다[Endres, 1975]. 202개의 모듈안에서 발견된 총 512개의 오유들가운데서 1개의 오유만이 112개의 모듈에서 발견되였다. 한편 일부 모듈들은 각각 14, 15, 19, 20개의 오유를 가진다는것이 밝혀 졌다. 엔드레스는 이 후자의 모듈에서 3개가 제품에서 가장 큰 모듈들이며 그 매개가 3,000행이상의 DOS마크로아쎔블리어명령을 포함하고 있

다는것을 지적하고 있다. 그리고 14개의 오유를 가지고 있는 모듈이 사전에 매우 불안정하다고 알려 진 상대적으로 크기가 작은 하나의 모듈이였다. 이런 류형의 모듈은 버리고 재코드작성하여야 할 첫 후보로 된다.

관리자측이 이러한 정황에 대처하기 위한 한가지 방도는 어떤 주어 진 모듈의 개발기간에 허용되는 오유의 최대개수를 사전에 결정하는것이다. 이 최대값에 도달하면 그 모듈을 버려야 하며 그다음 어떤 경험 있는 쏘프트웨어전문가가 그 모듈을 다시 설계하고 다시 코드작성하여야 한다. 이 최대값은 응용령역에 따라 또 모듈에 따라 변화될것이다. 결국 어떤 자료기지로부터 하나의 기록을 읽고 그 부분항목의 타당성을 검열하는 어떤 모듈에서 발견되는 오유의 최대허용개수는 각이한 수감기들에서 오는 자료들을 통합하고 대포의 과녁을 주어 진 목표에로 지향시키는 땅크의 무기계통조종체계안의 어떤 복잡한 모듈에서의 오유개수보다 훨씬 작아야 한다. 어떤 특정한 모듈에 대하여 최대오유합계를 결정하기 위한 한가지 방도는 오유교정유지정비를 요구한 류사한 모듈에서의 오유자료를 조사하는것이다. 그러나 어떤 평가기법이 리용되든지간에 관리자측은 이 합계가 초과되면 그 모듈은 페기된다는것을 확인하여야 한다(다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

어떤 모듈의 개발기간에 발견되는 오유의 최대허용개수와 관련한 론의는 명백히 개발기간에 허용되는 최대개수를 의미하고 있다. 제품이 의뢰자에게 배포된후에 발견되는 오유의 최대허용개수는 모든 제품의 모든 모듈에 대하여 령으로 되여야 한다. 즉 오류없는 코드를 의뢰자에게 배포하는것이 모든 쏘프트웨어공학자들의 목표로 되여야 한다.

14.15. 실현단계에서의 CASE도구

실현단계에서의 CASE도구들은 5.6에서 약간 자세히 론의되였다. 지면상의 리유로 하여 여기서는 그에 대하여 되풀이하지 않는다.

14.16. 항공음식전문회사 실례연구 : 검은통시험실례

그림 14-14는 항공음식전문회사제품을 위한 표본 검은통시험실례들을 보여 주고 있다. 완전한 실례모임은 부록 10에 제시된다. 시험실례들은 두가지 류형 즉 등가클라스와 경계값분석에 기초한 실례들과 기능시험에 기초한 실례들이다.

먼저 등가클라스와 경계값분석을 고찰하면 첫번째 모임의 다섯개 시험실례들은 등가클라스로부터 선택되여 어떤 승객의 세례명(firstName)이 1～15개의 문자로 구성되여 있는가를 검열한다. 다섯번째 등가클라스는 요구들에 만일 어떤 이름이 15개 문자이상으로 구성된다면 어떤 현상이 발생하는가를 서술하지 않고 있다는것을 반영하고 있다. 같은 모임안의 등가클라스들은 어떤 승객의 이름(lastName)에 마찬가지로 적용될수 있다.

등가클라스와 경계값분석

firstName나 lastName에 대한 등가클라스 :

1. <1개 문자 오유
2. 1개 문자 수용가능
3. 1～15개 문자 수용가능
4. 15개 문자 수용가능
5. >15개 문자 명시되지 않음

reservationID에 대한 등가클라스 :

1. <6개 문자 오유
2. >6개 문자 오유
3. 6개 문자(모두 자모가 아님) 오유
4. 6개 문자(모두 자모) 수용가능

기능분석

예약작성

1. reservationID가 이미 존재한다는 예약작성
2. 어떤 특별비행을 위한 좌석번호가 이미 예약된다는 예약작성
3. passengerID가 자료기지에 이미 존재한다는 예약작성
4. passengerID가 자료기지에 이미 존재하지 않는다는 예약작성

승객검사

5. reservationID를 자료기지에서 찾을수 없는 승객을 검사
6. reservationID를 자료기지에서 찾을수 있는 승객을 검사

특별식사목록의 조사

7. 자료기지에 존재하지 않는 비행날자와 비행기번호에 대하여 특별식사목록을 조사
8. 자료기지에 존재하는 비행날자와 비행기번호에 대하여 특별식사목록을 조사

우편엽서의 조사

9. reservationID를 자료기지에서 찾을수 없는 반환된 우편엽서를 조사
10. reservationID를 자료기지에서 찾을수 있는 반환된 우편엽서를 조사

보고서현시

11. 매 보고서에 대하여 출발날자(실례로 MAR/10/1995)가 마감날자(실례로 MAR/10/1990)보다 더 늦어 지는 보고서
12. 조달자와 탑승보고서에 대하여 자료기지에 존재하지 않는 비행날자와 비행기번호에 대한 보고서를 현시
13. 25h 조달자목록의 현시
14. 탑승식사목록의 현시
15. 주문한 식사가 한번이상 오르지 않은 승객이 없을 때 비적재보고서를 현시

그림 14-14. 항공음식전문회사 제품을 위한 검은통시험실례

16. 주문한 식사가 한번이상 오르지 않은 승객이 여러명일 때 비적재보고서를 현시 (같은 날자에 여러명의 예약이 발행할 때에도 우의 작용을 시도한다.)
17. 품질이 5급이하라고 생각되는 식사가 오른 경우의 승객이 자료기지안에 없을 때 저품질식사에 대한 보고서를 현시
18. 품질이 5급이하라고 생각되는 식사가 오른 경우의 승객이 자료기지안에 여러명일 때 저품질식사에 대한 보고서를 현시
19. 퍼센트에 대한 보고서를 현시
20. 자료기지에 저염식사항목이 없을 때 저염식사에 대한 보고서를 현시
21. 자료기지안에 저염식사항목이 여러개 있을 때 저염식사에 대한 보고서를 현시

그림 14-14. 항공음식전문회사제품을 위한 검은통시험실례(계속)

시험실례들이 명세서에서의 결함을 로출시킬 때 명세서에 되돌아 가 그것을 변경시키는것이 필요하다. 항공음식전문회사제품에서 명세서는 사용자가 15개 문자이상의 이름을 입력하려고 하면 어떤 오유통보가 인쇄되여야 한다는것을 서술하여야 한다. 검은통시험실례들에 의하여 로출된 명세서의 오유를 정정하기 위하여 필요한 변경조작은 단순하기때문에 여기서는 생략하기로 한다.

두번째 모임안의 4개의 등가클라스들은 예약식별자(*reservation*ID)가 정확히 6개의 자모문자들로 구성되여 있는가를 검열하는데 리용된다. 류사한 등가클라스들이 명세서안의 기타 설명문들을 검열하기 위한 시험실례들을 선택하는데 리용된다. 완전한 모임은 부록 10에 게시한다.

다음으로 기능시험을 고찰하자. 명세서로부터 끌어 낸 21개의 기능들을 그림 14-14에 렬거한다. 이 절에서 보여 주고 있는바와 같이 검은통시험실례들은 명세서안의 오유들을 밝혀 내는데 리용될수 있기때문에 명세서가 완성되자마자 곧 이러한 기능시험실례들이 개발되였다는것을 알아 차리는것이 중요하다.

매 시험계획에서 한가지 중요한 요소는 검은통시험실례들이 될수록 빨리 작성된다는데 대한 규정이다. 만일 제품전체에 대한 어떤 계약서가 서명된다면 이것은 명세서가 승인되자마자 곧 진행된다. 한편 제품의 완성에 관한 계약서는 일단 의뢰자가 기한과 비용평가에 동의하면 즉 명세서작성단계의 마감에 서명된다. 그다음 실현 및 통합단계에서 SQA그룹이 리용할수 있도록 검은통시험실례들이 작성된다.

14. 17. 실현단계에서의 난관

역설적으로 들릴지 모르지만 실현단계에서의 한가지 중요한 난관은 실현단계의 선행단계들에서 만족되여야 한다. 제8장에서 설명한바와 같이 코드의 재리용은 쏘프트웨어의 개발비용과 배포시간을 줄이기 위한 한가지 효과적인 방도로 된다. 그러나 이것은 실현단계처럼 늦게 시도하게 되면 코드의 재리용을 실현하기 어렵다.

실례로 언어 L로 어떤 제품을 실현하기로 결정된다고 하자. 이제 과반수의 모듈이 실현되고 시험된후에 관리자측이 쏘프트웨어제품의 도형사용자대면부를 위하여 프로그람

패키지 P를 리용하기로 결정한다고 하자. P의 루틴들이 아무리 강력하다 할지라도 만일 그 루틴들이 L와 결합하기 어려운 언어로 작성된다면 그 루틴들은 쏘프트웨어제품에서 재리용될수 없다.

비록 언어의 호상리용가능성이 문제로 되지 않는다고 하여도 재리용될 항목이 설계에 정확히 부합되지 않는 한 어떤 현존하는 모듈을 재리용하려고 할 때 일부 문제점이 존재하게 된다. 새로운 모듈을 처음부터 작성할 때보다 현존하는 모듈을 변경시키는데 더 많은 품이 요구될수 있다.

그러므로 모듈의 재리용은 맨 처음부터 쏘프트웨어제품안에 병합되여 들어 가야 한다. 재리용은 명세작성에 대한 제약으로뿐만아니라 사용자의 요구로 되여야 한다. 쏘프트웨어프로젝트관리계획(9.4)도 재리용을 병합하여야 한다. 또한 설계문서는 어느 모듈이 실현되며 어느 모듈이 재리용되는가를 서술하여야 한다.

결국 이 절의 서두에서 서술한바와 같이 비록 모듈의 재리용이 실현단계에서의 한가지 중요한 난관으로 된다고 하여도 모듈의 재리용은 요구사항확정, 명세서작성 및 설계단계들에 병합되여야 한다.

요 약

이 장에서는 개발팀에 의한 어떤 제품의 실현과 관계되는 여러가지 론의들을 제시하고 있다. 여기에는 프로그람작성언어의 선택문제가 포함된다(14.1). 4세대언어에 대한 문제가 14.2에서 약간 자세히 론의된다. 훌륭한 프로그람작성실천문제를 14.3에서 서술하며 실천적코드작성표준에 대한 요구가 14.4에 제시된다. 그다음 모듈의 재리용과 관련한 설명을 진행한다(14.5). 시험실례들은 체계적으로 선택되여야 한다는것이 서술된다(14.6). 검은통시험, 유리통시험, 비실행에 기초한 시험 등 여러가지 시험기법들이 서술되며 (14.7, 14.8, 14.9) 그다음 그 기법들을 비교한다(14.10). 무진실기법을 14.1.1에서 서술한다. 객체시험문제를 14.12에서 론의하며 뒤이어 모듈시험에 관한 관리측면문제가 론의된다(14.13). 모듈을 오유수정하지 않고 재작성하기 위한 문제를 14.14에서 서술한다. 실현단계에서의 CASE도구들을 그다음에 언급한다(14.15). 14.16에서 항공음식전문회사제품들에 대한 실례연구가 론의된다. 이 장은 실현단계에서의 난점에 대한 한가지 분석으로써 결속된다 (14.17).

보 충

4GL에 대한 보다 넓은 정보원천은 문헌 [Martin, 1985]이다. 4GL과 관련한 43개 기업체들의 립장은 문헌 [Guimaraes, 1985]에서 보고 하고 있다. 문헌 [Klepper and Bock, 1995]에는 McDonnell Douglas이 어떻게 3GL보다 4GL이 생산성이 더 높은가를 서술하고 있다.

좋은 프로그람작성실천과 관련한 훌륭한 문헌은 [Kernighan and Plauger, 1974, and McConnell, 1993] 이다.

실행에 기초한 시험과 관련된 보다 초기의 중요한 연구는 문헌 [Myers, 1979]이다. 일반적으로 시험에 대한 정보는 문헌 [Beizer, 1990]에 있다. 기능시험에 대하여서는 문헌 [Howden, 1987]에서 서술하여 주고 있고 구조적기법에 대하여서는 문헌 [Clarke, Podgurski, Richardson, and Zeil, 1989]에서 비교하여 주고 있다. 검은통시험에 대하여서는 문헌 [Beizer, 1995]에서 상세하게 서술하여 주었다. 검은통시험실례의 설계와 관련하여 문헌 [Yamaura, 1998]에서 서술하여 주었다. 기능적시험과 쏘프트웨어품질과 관련한 여러가지 피복측도들사이 관계에 대하여 문헌 [Horgan London, and Lyu, 1994]에서 론의하였다. 검은통시험에 관한 형식적인 연구방법에 대하여서는 문헌 [Stocks and Carrington, 1996]에서 론의하였다.

무진실과 관련하여서는 문헌 [Linger, 1994]에서 론의하였다. 유지정비단계에서의 무진실의 리용과 관련하여 문헌 [Sherer, Kouchakdjian, and Arnold, 1996]에서 서술하여 주었다. 무진실의 비평과 관련하여 문헌 [Beizer, 1997]에서 고찰하여 주었다.

쏘프트웨어믿음성에 대하여 문헌 [Musa and Everett, 1990]에서 소개하여 주고 있다. 그리고 쏘프트웨어믿음성공학과 관련한 국제토론회회보에는 쏘프트웨어믿음성과 관련한 아주 다양한 기사들을 게재하고 있다. 또한 믿음성에 대하여서는 *IEEE Software* 1995년 5월호와 *IEEE Computer* 1996년 11월호에 여러건의 기사들이 있다.

쏘프트웨어시험 및 분석과 관련한 국제토론회회보에는 시험과 관련한 여러가지 문제점들이 상세하게 서술되였다.

객체의 시험과 관련한 서로 다른 연구방법들의 조사에 대하여서는 문헌 [Turner, 1994]에서 찾아 볼수 있다. 이 주제에 대한 두개의 중요한 론문은 문헌 [Perry and Kaiser, 1990, and Harrold, McGregor, and Fitzpatrick, 1992]에 있다. 문헌 [Beizer, 1995]는 앞에서도 언급한바와 같이 객체지향쏘프트웨어에 대한 검은통시험을 포함하고 있다. *Communications of the ACM* 1994년 9월호에는 객체지향쏘프트웨어시험과 관련된 많은 기사들이 들어 있다.

실현단계에서의 척도와 관련한 맥케브의 주기적복잡도는 문헌 [McCabe, 1976]에서 처음으로 고찰하였다. 설계단계에서의 척도에 대한 확장은 문헌 [McCabe and Butler, 1989]에서 서술해 주었다. 쏘프트웨어과학과 주기적복잡도에 대한 타당성을 질문하는 기사들은 문헌 [Shepperd, 1988b; Weyuker, 1988b; and Shepperd and Ince, 1994]에 들어 있다. 코드검토를 관리하는 척도에 대하여서는 문헌 [Barnard and Price, 1994]에서 서술하여 주었다.

문 제

14.1. 교원이 당신에게 브로드랜즈지역 아동병원제품을 실현할것을 요청하였다(부록 1). 이 제품을 실현하기 위하여 당신은 어떤 언어를 선택하겠는가? 그 리유는 무엇인가?

당신이 리용할수 있는 여러가지 언어들에 대하여 그것들의 리득과 비용을 렬거하시오.

14.2. 승강기문제에 대하여 문제 14.1을 반복하시오(11.6.1).

14.3. 자동서고순환체계에 대하여 문제 14.1을 반복하시오(8.7).

14.4. 어떤 은행설명문의 정확성여부를 결정하는 제품에 대하여 문제 14.1을 반복하시오.

14.5. 자동출납기에 대하여 문제 14.1을 반복하시오(8.9).

14.6. 당신이 최근에 작성한 어떤 모듈에 서두설명문을 추가하시오.

14.7. 어떤 1인쏘프트웨어생산회사에 대한 코드작성표준은 300명의 쏘프트웨어전문가를 가진 개발기업체의 코드작성표준과 어떻게 차이나는가?

14.8. 집중치료기구를 위한 쏘프트웨어를 개발하고 유지정비하는 어떤 쏘프트웨어회사에 대한 코드작성표준은 회계제품을 개발하고 유지정비하는 개발기업체의 코드작성표준과 어떻게 차이나는가?

14.9. 나우르(Naur)의 본문처리문제(6.5.2)를 위한 검은통시험실례들을 작성하시오. 매 시험실례에 대하여 무엇이 시험되는가와 그 시험실례의 예상되는 결과를 서술하시오.

14.10. 문제 6.15에 대한 당신의 풀이(또는 지도교원이 배포한 코드)를 리용하여 명령문피복시험실례들을 작성하시오. 매 시험실례에 대하여 무엇이 시험되는가와 그 시험실례의 예상되는 결과를 서술하시오.

14.11. 분기피복에 대하여 문제 14.10을 반복하시오.

14.12. 정의와 리용사이의 모든 경로피복(*all-definition-use-path coverage*)에 대하여 문제 14.10을 반복하시오.

14.13. 경로피복에 대하여 문제 14.10을 반복하시오.

14.14. 선형코드순서렬에 대하여 문제 14.10을 반복하시오.

14.15. 문제 6.15에 대한 당신의 풀이(또는 지도교원이 배포하는 코드)에 대한 흐름도표를 작성하시오.

그 주기적복잡도를 결정하시오. 만일 분기수를 결정할수 없으면 흐름도를 하나의 방향그라프로서 고찰하시오. 릉의 개수 e, 마디점개수 n, 련결요소의 개수 c를 결정하시오(매 방법은 하나의 련결요소를 이룬다.). 주기적복잡도 V는 문헌 [McCabe, 1796]에 의해 다음과 같이 주어 진다.

$$V = e - n + 2c$$

14.16. 그림 14-10의 ㄴ)의 코드토막에 대한 할스테드(Halstead)의 4가지 기본척도를 결정하시오.

14.17. 당신은 1인쏘프트웨어회사의 주인이고 유일한 직원이다. 당신이 5.6에 서술된 프로그람작성도구집을 구입하였다. 중요한 순서로 5가지 능력을 렬거하고 그 리유를 밝

히시오.

14.18. 당신은 17,500명의 종업원을 가지고 있는 매우 큰 쏘프트웨어회사의 쏘프트웨어기술부의 부책임자이다. 5.6에서 서술한 프로그람작성도구집의 능력을 어떻게 평가하는가? 이 문제에 대한 당신의 대답과 선행한 대답사이의 모든 차이에 대하여 설명하시오.

14.19. 당신은 쏘프트웨어개발기업체를 위한 SQA경영자로서 시험기간에 어떤 주어진 모듈에서 발견할수 있는 최대오유개수를 결정할 책임을 지고 있다. 만일 이 최대값이 초과되면 그 모듈은 다시 설계되고 다시 코드작성되여야 한다. 어떤 판정기준을 리용하여 주어 진 모듈에 대한 최대오유개수를 결정하겠는가?

14.20. (과정안상 목표) 문제 11.15 또는 12.9에 명시된 제품을 위한 검은통시험실례들을 작성하시오. 매 시험실례에 대하여 무엇이 시험되는가와 그 시험실례의 예상되는 결과를 서술하시오.

14.21. (실례연구) 15.13에 서술한 항공음식전문회사제품의 실례연구실현에 대한 복제본이 주어 진다. 이 제품에 대한 명령문피복시험실례를 작성하시오. 매 시험실례에 대하여 무엇이 시험되는가와 그 시험실례의 예상되는 결과를 서술하시오.

14.22. (실례연구) 분기피복에 대하여 문제 14.21을 반복하시오.

14.23. (실례연구) 정의와 리용사이의 모든 경로피복에 대하여 문제 14.21을 반복하시오.

14.24. (실례연구) 경로피복에 대하여 문제 14.21을 반복하시오.

14.25. (실례연구) 선행코드순서렬에 대하여 문제 14.21을 반복하시오.

14.26. (쏘프트웨어공학독본) 교원이 문헌 [Beizer, 1997]의 복제본을 배포할것이다. 무진실에 대한 당신의 립장은 무엇인가?

참 고 문 헌

[Barnard and Price, 1994] J. Barnard and A. Price, "Managing Code Inspection Information," *IEEE Software* **11** (March 1994), pp. 59–69.

[Basili and Hutchens, 1983] V. R. Basili and D. H. Hutchens, "An Empirical Study of a Syntactic Complexity Family," *IEEE Transactions on Software Engineering* **SE-9** (November 1983), pp. 664–672.

[Basili and Selby, 1987] V. R. Basili and R. W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering* **SE-13** (December 1987), pp. 1278–96.

[Basili and Weiss, 1984] V. R. Basili and D. M. Weiss, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering* **SE-10** (November 1984), pp. 728–38.

[Beizer, 1990] B. Beizer, *Software Testing Techniques,* 2nd ed., Van Nostrand Reinhold, New York, 1990.

[Beizer, 1995] B. Beizer, *Black-Box Testing: Techniques for Functional Testing of Software and Systems,* John Wiley and Sons, New York, 1995.

[Beizer, 1997] B. Beizer, "Cleanroom Process Model: A Critical Examination," *IEEE Software* **14** (March/April 1997), pp. 14–16.

[Clarke, Podgurski, Richardson, and Zeil, 1989] L. A. Clarke, A. Podgurski, D. J. Richardson, and S. J. Zeil, "A Formal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions on Software Engineering* **15** (November 1989), pp. 1318–32.

[Crossman, 1982] T. D. Crossman, "Inspection Teams, Are They Worth It?" *Proceedings of the Second National Symposium on EDP Quality Assurance,* Chicago, November 1982.

[Date, 1999] C. J. Date, *An Introduction to Database Systems,* 7th ed., Addison-Wesley, Reading, MA, 1999.

[Dunn, 1984] R. H. Dunn, *Software Defect Removal,* McGraw-Hill, New York, 1984.

[Endres, 1975] A. Endres, "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering* **SE-1** (June 1975), pp. 140–49.

[Grady, 1992] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement,* Prentice Hall, Englewood Cliffs, NJ, 1992.

[Guimaraes, 1985] T. Guimaraes, "A Study of Application Program Development Techniques," *Communications of the ACM* **28** (May 1985), pp. 494–99.

[Halstead, 1977] M. H. Halstead, *Elements of Software Science,* Elsevier North-Holland, New York, 1977.

[Harrold, McGregor, and Fitzpatrick, 1992] M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick, "Incremental Testing of Object-Oriented Class Structures," *Proceedings of the 14th International Conference on Software Engineering,* Melbourne, Australia, May 1992, pp. 68–80.

[Horgan, London, and Lyu, 1994] J. R. Horgan, S. London, and M. R. Lyu, "Achieving Software Quality with Testing Coverage Measures," *IEEE Computer* **27** (1994), pp. 60–69.

[Howden, 1987] W. E. Howden, *Functional Program Testing and Analysis,* McGraw-Hill, New York, 1987.

[Hwang, 1981] S.-S. V. Hwang, "An Empirical Study in Functional Testing, Structural Testing, and Code Reading Inspection," Scholarly Paper 362, Department of Computer Science, University of Maryland, College Park, 1981.

[Kernighan and Plauger, 1974] B. W. Kernighan and P. J. Plauger, *The Elements of Programming Style,* McGraw-Hill, New York, 1974.

[Klepper and Bock, 1995] R. KLEPPER AND D. BOCK, "Third and Fourth Generation Productivity Differences," *Communications of the ACM* **38** (September 1995), pp. 69–79.

[Klunder, 1988] D. KLUNDER, "Hungarian Naming Conventions," Technical Report, Microsoft Corporation, Redmond, WA, January 1988.

[Linger, 1994] R. C. LINGER, "Cleanroom Process Model," *IEEE Software* **11** (March 1994), pp. 50–58.

[Martin, 1985] J. MARTIN, *Fourth-Generation Languages*, Volumes 1, 2, and 3, Prentice Hall, Englewood Cliffs, NJ, 1985.

[McCabe, 1976] T. J. MCCABE, "A Complexity Measure," *IEEE Transactions on Software Engineering* **SE-2** (December 1976), pp. 308–20.

[McCabe and Butler, 1989] T. J. MCCABE AND C. W. BUTLER, "Design Complexity Measurement and Testing," *Communications of the ACM* **32** (December 1989), pp. 1415–25.

[McConnell, 1993] S. MCCONNELL, *Code Complete: A Practical Handbook of Software Construction*, Microsoft Press, Redmond, WA, 1993.

[Mills, Dyer, and Linger, 1987] H. D. MILLS, M. DYER, AND R. C. LINGER, "Cleanroom Software Engineering," *IEEE Software* **4** (September 1987), pp. 19–25.

[Musa and Everett, 1990] J. D. MUSA AND W. W. EVERETT, "Software-Reliability Engineering: Technology for the 1990s," *IEEE Software* **7** (November 1990), pp. 36–43.

[Musa, Iannino, and Okumoto, 1987] J. D. MUSA, A. IANNINO, AND K. OKUMOTO, *Software Reliability: Measurement, Prediction, Application*, McGraw-Hill, New York, 1987.

[Myers, 1976] G. J. MYERS, *Software Reliability: Principles and Practices*, Wiley-Interscience, New York, 1976.

[Myers, 1978a] G. J. MYERS, "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," *Communications of the ACM* **21** (September 1978), pp. 760–68.

[Myers, 1979] G. J. MYERS, *The Art of Software Testing*, John Wiley and Sons, New York, 1979.

[Ottenstein, 1979] L. M. OTTENSTEIN, "Quantitative Estimates of Debugging Requirements," *IEEE Transactions on Software Engineering* **SE-5** (September 1979), pp. 504–14.

[Perry and Kaiser, 1990] D. E. PERRY AND G. E. KAISER, "Adequate Testing and Object-Oriented Programming," *Journal of Object-Oriented Programming* **2** (January/February 1990), pp. 13–19.

[Rapps and Weyuker, 1985] S. RAPPS AND E. J. WEYUKER, "Selecting Software Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering* **SE-11** (April 1985), pp. 367–75.

[Sammet, 1978] J. E. SAMMET, "The Early History of COBOL," *Proceedings of the History of Programming Languages Conference*, Los Angeles, 1978, pp. 199–276.

[Shepperd, 1988b] M. SHEPPERD, "A Critique of Cyclomatic Complexity as a Software Metric," *Software Engineering Journal* **3** (March 1988), pp. 30–36.

[Shepperd and Ince, 1994] M. SHEPPERD AND D. C. INCE, "A Critique of Three Metrics," *Journal of Systems and Software* **26** (September 1994), pp. 197–210.

[Sherer, Kouchakdjian, and Arnold, 1996] S. W. SHERER, A. KOUCHAKDJIAN, AND P. G. ARNOLD, "Experience Using Cleanroom Software Engineering," *IEEE Software* **13** (May 1996), pp. 69–76.

[Stocks and Carrington, 1996] P. STOCKS AND D. CARRINGTON, "A Framework for Specification-Based Testing," *IEEE Transactions on Software Engineering* **22** (November 1996), pp. 777–93.

[Takahashi and Kamayachi, 1985] M. TAKAHASHI AND Y. KAMAYACHI, "An Empirical Study of a Model for Program Error Prediction," *Proceedings of the Eighth International Conference on Software Engineering,* London, 1985, pp. 330–36.

[Trammel, Binder, and Snyder, 1992] C. J. TRAMMEL, L. H. BINDER, AND C. E. SNYDER, "The Automated Production Control Documentation System: A Case Study in Cleanroom Software Engineering," *ACM Transactions on Software Engineering and Methodology* **1** (January 1992), pp. 81–94.

[Turner, 1994] C. D. TURNER, "State-Based Testing: A New Method for the Testing of Object-Oriented Programs," Ph.D. thesis, Computer Science Division, University of Durham, Durham, U.K., November, 1994.

[Walsh, 1979] T. J. WALSH, "A Software Reliability Study Using a Complexity Measure," *Proceedings of the AFIPS National Computer Conference,* New York, 1979, pp. 761–68.

[Watson and McCabe, 1996] A. H. WATSON AND T. J. MCCABE, "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric," NIST Special Publication 500–235, Computer Systems Laboratory, National Institute of Standards and Technology, Gaithersburg, MD, 1996.

[Weyuker, 1988a] E. J. WEYUKER, "An Empirical Study of the Complexity of Data Flow Testing," *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis,* Banff, Canada, July 1988, pp. 188–95.

[Weyuker, 1988b] E. WEYUKER, "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering* **14** (September 1988), pp. 1357–65.

[Wilde, Matthews, and Huitt, 1993] N. WILDE, P. MATTHEWS, AND R. HUITT, "Maintaining Object-Oriented Software," *IEEE Software* **10** (January 1993), pp. 75–80.

[Woodward, Hedley, and Hennell, 1980] M. R. WOODWARD, D. HEDLEY, AND M. A. HENNELL, "Experience with Path Analysis and Testing of Programs," *IEEE Transactions on Software Engineering* **SE-6** (May 1980), pp. 278–86.

[Yamaura, 1998] T. YAMAURA, "How to Design Practical Test Cases," *IEEE Software* **15** (November/December 1998), pp. 30–36.

제15장. 실현 및 통합단계

지금까지 실현과 통합은 두개의 독립적이고 분리된 단계로 취급되여 왔다. 제시된 방법들에서 매개의 모듈(또는 객체)은 프로그람작성팀의 한 성원에 의하여 실현되고 쏘프트웨어품질보증그룹에 의하여 시험된다. 그다음 통합단계에서 모듈들이 합쳐 지고 전체로서 시험된다. 사실상 이 방법은 쏘프트웨어를 개발하기 위한 좋은 방법으로 되지 못한다. 그대신 실현과 통합단계는 병렬로 수행되여야 한다. 병렬적방법이 이 장의 기본주제로 된다.

이 장에서는 실현 및 통합단계(*implementation and integration phase*)에 대하여 언급한다. 엄격히 말하여 이와 같은 단계는 없다. 실현과 통합은 설계와 계획이 구별되는것과 마찬가지로 서로 다른 활동이다. 그러나 지금부터 실현 및 통합단계라는 용어를 리용하여 설계단계와 통합단계가 병렬로 진행될 때에 수행되는 활동을 서술한다.

여기에서는 통합 및 통합시험에 관한 기본개념들이 소개되며 다음으로 객체지향제품들이 어떻게 실현되고 통합되는가를 서술한다.

15.1. 실현 및 통합에 대한 소개

그림 15-1에 묘사된 제품을 고찰하자. 이 제품을 개발할 때의 한가지 방법은 매개의 모듈은 개별적으로 코드작성하고 시험하며 그다음 13개의 모듈을 모두 결합하고 전체적인 제품을 시험하는것이다. 이 방법에는 두가지 난점이 있다. 첫째로, 모듈 a를 고찰하자. 모듈 a는 모듈 b, c, d를 호출하기때문에 자기자신에 대하여 시험될수 없다. 그러므로 모듈 a를 시험하기 위하여서는 모듈 b, c, d는 대용체(*stub*)로서 코드작성되여야 한다. 가장 단순한 형식으로서 하나의 대용체는 하나의 빈 모듈이다. 보다 유효한 하나의 대용체는 module displayRadarPattern called와 같은 하나의 통보를 인쇄한다.

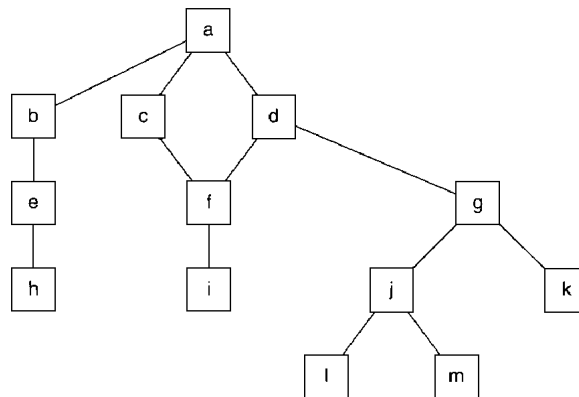



그림 15-1. 전형적인 모듈호상접속도

무엇보다도 하나의 대용체는 사전에 계획된 시험실례들에 대응하는 값들을 귀환하여야 한다. 이제 모듈 h를 고찰하자. 모듈 h를 자기자신에 대하여 시험하기 위하여서는 어떤 구동프로그람 즉 만일 시험하는 모듈이 귀환하는 값들을 검열할수 있다면 그 모듈을 한번 또는 그이상 호출하는 하나의 모듈이 필요하다. 이와 류사하게 모듈 d를 시험하려면 하나의 구동프로그람과 두개의 대용체가 필요하다. 그러므로 실현과 통합을 분리하여 진행할 때 발생하는 한가지 문제는 모듈시험이 완성된후에 모두 버리게 되는 대용체들과 구동프로그람을 작성하는데 품을 들여야 한다는것이다.

실현단계가 완성된 다음 통합단계가 시작될 때 발생하는 보다 중요한 두번째 난점은 오유의 고립이 결여되여 있는것이다. 만일 전체적인 제품이 어떤 특정한 실례에 대하여 시험되여 실패로 된다면 오유는 13개의 모듈 또는 13개의 대면부중에서 임의의것에 존재할수 있다. 어떤 보다 큰 제품 이를테면 103개의 모듈과 108개의 대면부를 가진 제품에서는 오유가 존재할수 있는 개소가 211개이상이다.

이 두가지 난점에 대한 해결방안은 모듈시험과 통합시험를 결합하는것이다.

15. 1. 1. 내리실현 및 통합

내리실현 및 통합에서 만일 모듈 mAbove가 모듈 mBelow를 호출한다면 mAbove는 mBelow에 앞서 실현되고 통합된다. 그림 15-1에 보여 준 제품이 하강식으로 실현되고 통합된다고 하자. 한가지 가능한 하강식순서짓기는 a, b, c, d, e, f, g, h, i, j, k, l, m이다. 먼저 모듈 a가 코드작성되고 대용체로 실현된 b, c, d와 함께 시험된다. 그다음 대용체 b는 모듈 b로 확장되여 모듈 a와 결합되며 대용체로 실현된 모듈 e와 함께 시험된다. 실현 및 통합은 이러한 방식으로 모든 모듈들이 제품안에 통합되여 들어갈 때까지 진행된다. 또 한가지 가능한 하강식순서짓기는 a, b, e, h, c, d, f, i, g, j, k, l, m이다. 이 순서짓기에 대하여 실현 및 통합의 부분들은 다음과 같은 방식으로 병렬로 진행된다. 모듈 a가 코드작성되고 시험된 다음에 한명의 프로그람작성자가 모듈 a를 리용하여 b, e, h를 실현하고 통합할수 있으며 한편 다른 프로그람작성자가 a를 리용하여 c, d, f, i에 대하여 병렬로 작업할수 있다. 일단 d와 f가 완성되면 세번째 프로그람작성자가 g, j, k, l, m에 대하여 작업을 시작할수 있다.

모듈 a가 그자체로서 어떤 특정한 시험실례에 대하여 정확하게 실행된다고 가정하자. 그러나 b가 코드작성되여 제품에 통합되여 들어 가면 그 제품은 모듈 a와 b가 서로 결합되여 이루어 지는데 그다음 같은 시험자료가 제시되는 경우 그 제품은 실패한다. 오유는 두 개소중에서 어느 하나에 있을수 있다. 즉 모듈 b든가 또는 모듈 a와 b사이의 대면부에 있을수 있다. 일반적으로 어떤 모듈 mNew가 지금까지 시험된것들에 추가되여 사전에 성공하였던 어떤 시험실례가 실패하는 경우에는 언제나 오유는 거의 확정적으로 mNew든가 mNew와 제품의 기타 부분사이의 대면부들에 놓이게 된다. 결국 내리실현 및 통합은 오유고립화를 지원한다.

내리실현 및 통합의 다른 우점은 기본설계오유들이 일찌기 드러나게 된다는것이다. 어떤 제품의 모듈들은 두개의 그룹 즉 론리모듈과 조작모듈로 가를수 있다. 론리모듈

(*logic module*)은 본질에 있어서 제품의 조종측면에 대한 결심채택흐름을 병합하고 있다. 론리모듈들은 일반적으로 모듈호상접속도에서 뿌리에 가깝게 배치되여 있는 모듈들이다. 실례로 그림 15-1에서 a, b, c, d를 론리모듈로 기대하는것이 타당하며 g와 j도 론리모듈로 될수 있을것 같다. 한편 조작모듈(*operational module*)은 제품의 실제적인 조작을 수행한다. 실례로 하나의 조작모듈은 getlineFromTerminal이든가 measureTemperatureOfReactorCore로 이름지어 질수 있다. 조작모듈들은 일반적으로 모듈호상접속도의 잎에 가까운 보다 낮은 준위에서 찾아 진다. 그림 15-1에서 모듈 e, f, h, i, k, l, m은 조작모듈들이다.

조작모듈들을 코드작성하고 시험하기전에 론리모듈들을 코드작성하고 시험하는것이 중요하다. 이것은 임의의 중요한 설계오유들이 일찌기 로출되게 할것이다. 전체 제품이 완성된 다음 어떤 중요한 오유가 발견된다고 가정하자. 그러면 제품의 많은 부분을 재작성하여야 할것이다. 특히 조종흐름을 구현하고 있는 론리모듈을 재작성하여야 한다. 아마도 대부분의 조작모듈들은 재구성된 제품내에서 재리용가능할수 있다. 실례로 getlineFromTerminal 또는 measureTemperatureOfReactorCore와 같은 모듈은 그 제품이 어떻게 재구성된다고 하더라도 필요하다. 그러나 조작모듈이 제품의 다른 모듈과 련결되는 방법은 변경되여야 할수 있으며 이것은 불필요한 작업을 야기시킨다. 그러므로 설계오유가 일찌기 발견되면 될수록 그 제품을 정정하고 늦어 진 개발일정을 보상하는것이 더욱 빨라 지고 비용이 더 적게 든다. 모듈들이 하강식전략을 리용하여 실현되고 통합되는 순서는 본질에 있어서 론리모듈들이 실지로 조작모듈보다 앞서 실현되고 통합된다는것을 담보한다. 왜냐하면 모듈호상접속도에서 론리모듈은 거의 언제나 조작모듈의 선조로 되기때문이다.

그러나 내리실현 및 통합은 한가지 결함을 가지고 있다. 즉 잠재적으로 재리용가능한 모듈들이 적당하게 시험되지 않을수 있다. 철저히 시험되였다고 부당하게 인정된 어떤 모듈을 재리용하는것은 그 모듈을 처음부터 작성하는것보다 더 비경제적일수 있다. 왜냐하면 어떤 모듈이 정확하다는 가정은 그 모듈이 실패할 때 그릇된 결론에로 이끌어 갈수 있기때문이다. 즉 불충분하게 시험되여 재리용된 모듈을 의심할 대신에 시험자는 오유가 그밖의 다른데 있다고 생각하면서 결국은 노력을 랑비할수 있다.

론리모듈은 특정한 일부 문제에 한정되여서 다른 련관범위에서는 리용할수 없을것 같다. 그러나 조작모듈은 특히 그것이 비형식적응집도를 가지는 경우에(7.2.7) 미래의 제품들에 재리용될수 있으며 그때문에 철저한 시험을 필요로 한다. 유감스럽게도 조작모듈은 일반적으로 모듈호상접속도에서 하위준위모듈이며 따라서 상위준위모듈처럼 자주 시험되지 않는다. 실례로 184개의 모듈이 있다면 뿌리모듈은 184번 시험될것이며 반면에 제품에 마지막으로 통합된 모듈은 한번만 시험될것이다. 내리실현 및 통합은 조작모듈에 대한 부적당한 시험으로 인하여 재리용을 위험한 일로 만든다.

만일 제품이 잘 설계되면 정황은 더욱 악화된다. 사실 설계가 좋으면 좋을수록 모듈들은 더 불철저하게 시험되는것 같다. 이것을 보기 위하여 모듈 computeSquareRoot를 고찰하자. 이 모듈은 두개의 인수를 가지고 있다. 즉 2차뿌리로 결정해야 할 류점수 x와 x가 부수일 때 true로 설정되는 errorFlag이다. 나아가서 computeSquareRoot가 모듈 m3에 의하여 호출되며 모듈 m3은 다음의 명령문을 포함하고 있다고 가정하자.

```
If  (x>=0)
        y=computeSquareRoot(x,  errorFlag);
```

달리 말하면 computeSquareRoot는 x의 값이 부수가 아닌 이상 절대로 호출되지 않으며 그러므로 이 모듈은 부의 x값으로는 절대로 시험되지 않기때문에 그것이 정확하게 기능을 수행하는가를 알수 없다. 모듈호출이 이러한 종류의 어떤 안정성시험을 포함하는 설계형식을 방어프로그람작성(*defensive programming*)이라고 부른다. 방어프로그람작성의 결과로 만일 모듈들이 하강식으로 실현되고 통합된다면 종속조작모듈들은 철저하게 시험되지 않을것 같다. 방어프로그람작성에 대한 한가지 대안은 믿음성구동설계를 리용하는 것이다(1.6). 여기에서는 필요한 안정성시험이 호출자가 아니라 호출된 모듈안에 작성되여 들어 간다. 다른 하나의 방법은 호출된 모듈에서 단언들을 리용하는것이다(6.5.3).

15. 1. 2. 올리실현 및 통합

올리실현 및 통합에서 만일 모듈 mAbove가 모듈 mBelow를 호출한다면 mBelow는 mAbove보다 앞서 실현되고 통합된다. 그림 15-1에서 가능한 한가지 상승식순서짓기는 1, m, h, i, j, k, e, f, g, b, c, d, a이다. 이 제품을 어떤 팀이 코드작성하도록 하자면 보다 좋은 한가지 상승식순서짓기는 다음과 같이 진행된다. h, e, b가 한 프로그람작성자에게 주어 지며 i, f, c는 다른 프로그람작성자에게 주어 진다. 세번째 프로그람작성자는 1, m, j, k, g로써 시작하여 그다음 d를 실현하고 자기의 작업결과를 두번째 프로그람작성자의 작업결과와 통합한다. 마지막으로 b, c, d가 성공적으로 통합될 때 a는 실현되고 통합될수 있다.

결국 조작모듈들은 상승식전략이 리용될 때 철저히 시험된다. 이밖에 시험은 호출모듈이 방어적으로 프로그람작성되는 오유차페에 의해서가 아니라 구동프로그람의 도움으로 수행된다. 비록 올리실현 및 통합이 내리실현 및 통합의 기본난점을 해결하고 내리실현 및 통합의 우점인 오유고립화를 공유한다고 하여도 그것 역시 자기의 난점을 가지고 있다. 보다 명확하게 말하면 중요한 설계오유들이 통합단계에서 늦게 발견된다. 론리모듈들은 마지막에 통합된다. 때문에 만일 어떤 중요한 설계오유가 존재하면 그 오유는 그 제품의 많은 부분을 재설계하고 재코드작성하는데 막대한 비용을 소비한 결과로 통합단계의 마지막에야 발견될것이다.

그러므로 하강식과 상승식설계 및 통합은 모두 자기의 우점과 약점을 가지고 있다. 제품개발을 위한 해결방안은 우점을 리용하고 약점은 최소화하는 방식으로써 두 전략을 결합하는것이다. 이것은 쌘드위치식실현 및 통합에 관한 착상을 가져 왔다.

15. 1. 3. 쌘드위치식실현 및 통합

그림 15-2에 보여 준 모듈호상접속도를 고찰하자. 6개의 모듈 a, b, c, d, g, j들은 론리모듈이며 그러므로 이것들은 하강식으로 실현되고 통합되여야 한다. 7개의 모듈 e, f, h, i, k, l, m들은 조작모듈이며 그것들은 상승식으로 실현되고 통합되여야 한다. 즉 하강식

도 상승식도 모든 모듈에 적당하지는 않기때문에 해결방안은 그것들을 부분별로 가르는 것이다. 6개의 론리모듈들은 하강식으로 실현되고 통합되며 중요한 모든 설계오유들은 일찌기 발견될수 있다. 7개의 조작모듈들은 상승식으로 실현되고 통합된다. 결국 이 모듈들은 방어적으로 프로그람작성된 호출모듈들에 의하여 차페되지 않으면서 어떤 철저한 시험을 받으며 그때문에 다른 제품들에 대한 확신과 더불어 재리용될수 있다. 모든 모듈들이 적당하게 통합되였을 때 두 모듈그룹사이의 대면부들이 하나씩 시험된다. 쌘드위치식실현 및 통합(*sandwich implementation and integration*)이라고 부르는 이 과정이 수행되는 전 기간 오유고립화가 만족된다(다음의 《알고 싶은 문제》를 보시오.). 그림 15-3은 상승식과 내리실현 및 통합뿐만아니라 쌘드위치식실현 및 통합의 우점과 약점을 종합하고 있다.

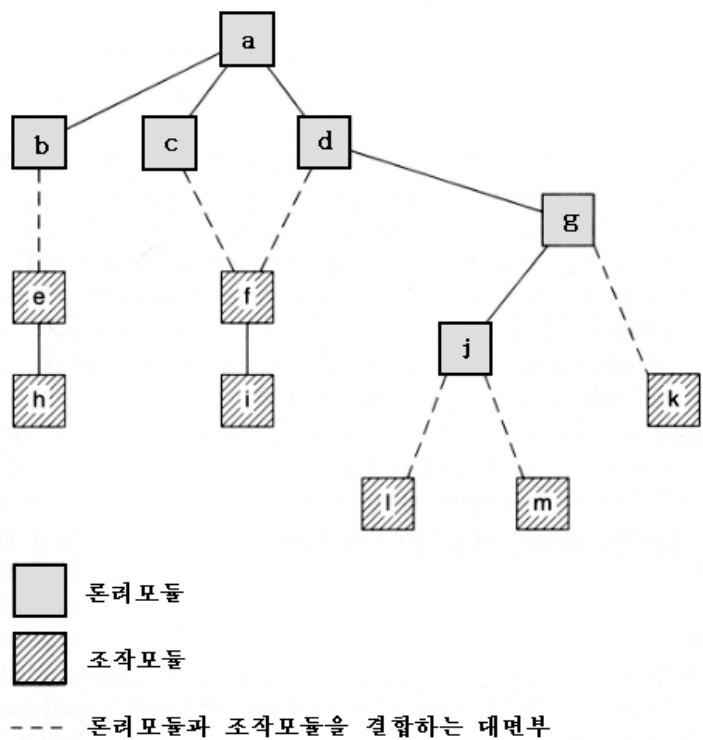



그림 15-2. 쌘드위치식실현 및 통합을 리용하여 개발된 그림 15-1의 제품

알고 싶은 문제

용어 쌘드위치식실현 및 통합(*sandwich implementation and integration*) [Myers, 979]은 론리모듈과 조작모듈을 쌘드위치의 웃부분과 아래부분으로, 그것들을 련결하는 대면부를 쌘드위치의 속으로 간주한데로부터 유래되였다. 이것을 그림 15-2에서 볼수 있다.

연구방법	우　점	약　점
실현후 통합(15.1)	—	오유고립화가 실현되지 않는다. 기본설계오유들이 늦게 로출된다.
내리실현과 통합(15.1.1)	오유고립화를 실현한다. 기본설계오유는 일찌기 로출된다.	잠재적으로 재리용가능한 모듈들이 충분히 시험되지 않는다.
올리실현과 통합(15.1.2)	오유고립화를 실현한다. 잠재적으로 재리용가능한 모듈들이 충분히 시험된다.	기본설계오유들이 늦게 로출된다.
쌘드위치식실현과 통합(15.1.3)	오유고립화를 실현한다. 기본설계오유는 일찌기 로출된다. 잠재적으로 재리용가능한 모듈들이 충분히 시험된다.	—

그림 15-3. 실현 및 통합방법의 개괄과 서술한 절

쌘드위치식실현 및 통합을 아래의 란에 종합한다.

쌘드위치식실현 및 통합을 진행하는 방법

다음의 조작을 병렬로 진행한다.
- 론리모듈들을 하강식으로 실현하고 통합한다.
- 조작모듈들을 상승식으로 실현하고 통합한다.

다음조작을 진행한다.
- 론리모듈과 조작모듈사이의 대면부를 시험한다.

15. 1. 4. 객체지향제품의 실현 및 통합

객체들은 상승식으로 또는 하강식으로 실현되고 통합된다. 만일 내리실현 및 통합이 선택된다면 대용체들은 고전적인 모듈들과 같은 방식으로 매개의 방법에 리용된다.

만일 올리실현 및 통합이 리용된다면 다른 객체들에 통보를 보내지 않는 객체들이 먼저 실현되고 통합된다. 그다음 이 객체들에 통보를 보내는 객체들이 실현되고 통합되는데 이 과정은 제품안의 모든 객체들이 실현되고 통합될 때까지 계속된다(이 과정은 재귀가 존재하는 경우에는 변경되여야 한다.).

하강식과 올리실현 및 통합이 모두 지원되는것으로 하여 쌘드위치식실현 및 통합도

역시 리용될수 있다. 만일 제품이 C++와 같은 혼성식객체지향언어로 실현된다면 객체들은 흔히 고전적파라다임의 조작모듈들에 대응하며 이로부터 상승식으로 실현되고 통합된다. 객체들이 아닌 대부분의 모듈들은 론리모듈로 된다. 이 모듈들은 하강식으로 실현되고 통합된다. 다른 모듈들은 조작모듈로 되며 따라서 그것들은 상승식으로 실현되고 통합된다. 결국 모든 비객체모듈들은 객체들과 통합된다.

제품이 Java와 같은 순수한 객체지향언어를 리용하여 실현될 때에 조차도 main과 같은 클라스방법들(때로는 정적방법(*static method*)이라고 부른다.)과 활용방법들은 일반적으로 구조화파라다임의 론리모듈들과 구조상 류사하다. 그러므로 클라스방법들 역시 하강식으로 실현되며 그다음 다른 객체들과 통합된다. 달리 말하면 하나의 객체지향제품을 실현하고 통합할 때 쌘드위치식실현 및 통합의 변종들이 리용된다.

15. 1. 5. 실현 및 통합단계에서의 관리문제

관리와 관련한 한가지 문제는 통합할 때에 일부 모듈들이 단순하게 서로 결합되지 않는다는것을 발견하는것이다. 실례로 프로그람작성자 1은 객체 o1을 코드작성하고 프로그람작성자 2는 객체 o2를 코드작성한다고 하자. 프로그람작성자 1이 리용하는 설계문서에서는 객체 o1이 객체 o2에 4개의 인수를 통해 어떤 통보를 보내지만 프로그람작성자 2가 리용하는 설계문서에서는 다만 3개의 인수만이 o2에 보내진다는것을 명백히 서술하고 있다. 이와 같은 문제는 어떤 변경이 개발그룹의 모든 성원들에게 통보되지 않고 설계문서의 하나의 복제본에서만 만들어 지는 경우에 발생할수 있다. 두 프로그람작성자는 모두 자기가 옳다고 생각하며 서로 타협하려고 하지 않는다. 왜냐하면 양보하는 프로그람작성자는 그 제품의 많은 부분을 재코드작성하여야 하기때문이다.

이러한 문제들과 이와 류사한 불일치문제들을 해결하기 위하여서는 전체 통합과정을 SQA그룹이 실행하도록 하여야 한다. 더우기 다른 단계에서의 시험과 마찬가지로 SQA그룹은 통합시험이 부적당하게 진행된다면 큰 손실을 입게 된다. 그러므로 SQA그룹은 시험이 철저히 진행되도록 할 가능성이 매우 크다. 때문에 SQA그룹의 관리자는 통합단계의 모든 측면들에 대하여 책임을 져야 한다. 즉 관리자는 어느 모듈을 하강식으로 실현하고 통합하며 어느 모듈은 상승식으로 실현하고 통합하겠는가를 결정하여야 하며 또 통합시험과제들을 적당한 개별적사람들에게 할당해 주어야 한다. 쏘프트웨어제품개발계획에서 통합시험계획을 작성하게 될 SQA그룹은 그 계획의 실현에 대하여 책임을 진다.

통합단계의 마감에 모든 모듈들은 시험되여 단일한 제품으로 결합되게 된다.

15. 2. 실현 및 통합단계에서의 시험

여러가지 각이한 시험이 실현 및 통합단계에서 진행되여야 한다. 우선 새로운 매개의 모듈은 그것이 이미 통합된 부분에 추가될 때 시험되여야 한다. 여기서 중요한 점은 새로운 모듈을 제14장에서 서술한것처럼 시험하고 부분제품의 나머지부분들이

새로운 모듈이 통합되기전과 마찬가지로 계속 동작한다는것을 확인하는것이다.

제품이 도형사용자대면부를 가질 때에는 통합시험과 관련하여 특수한 문제점들이 발생할수 있다. 이에 대하여 다음 절에서 론의한다.

15. 3. 도형사용자대면부의 통합시험

어떤 제품을 시험하는것은 어떤 파일안의 어떤 시험실례에 대하여 입력자료들을 정렬함으로써 간단하게 될수 있다. 그다음 그 제품이 실행되고 련관된 자료들이 그에 따라 실행된다. 비교적 초보적인 어떤 CASE도구의 도움으로 전체 공정이 자동화될수 있다. 즉 어떤 시험실례들의 모임이 매 실례의 예상되는 결과와 함께 작성된다. CASE도구는 매 시험실례들을 실행하여 실지결과를 예상되는 결과와 비교하며 사용자에게 매 실례에 대하여 보고한다. 그다음 시험실례들은 그 제품이 변경될 때마다 진행하는 회귀시험에 리용하기 위하여 보관된다. SilkTest는 이러한 도구의 한가지 실례로 된다.

그러나 어떤 제품이 어떤 도형사용자대면부를 병합하는 경우에는 이 방법을 적용할수 없다. 특히 어떤 차림표를 내리 펼치거나 마우스로 찰칵하기 위한 시험자료는 일반적인 시험자료와 같은 방식으로 파일안에 기억될수 없다. 동시에 GUI를 수동적으로 시험하는것은 시간이 걸리고 지루한 작업이다. 이 문제에 대한 한가지 해결방안은 마우스찰칵, 건반누르기 등과 같은 기능을 유지하는 어떤 특수한 CASE도구를 리용하는것이다. GUI는 CASE도구가 시험파일을 작성할수 있도록 일단 수동적으로 시험된다. 그다음부터 이 파일은 그이후의 자료들에 리용된다. QAPartner와 XRunner를 비롯한 여러가지 CASE도구들이 GUI의 시험을 지원하고 있다.

통합과정이 완성될 때 제품전체가 시험된다. 이것을 제품시험(*product testing*)이라고 부른다. 개발자들이 제품의 모든 측면의 정확성에 대하여 확신하게 될 때 그 제품은 인수시험(*acceptance testing*)을 위하여 의뢰자에게 넘겨 진다. 이 두가지 류형의 시험을 보다 상세히 론의한다.

15. 4. 제품시험

마지막모듈들이 제품안에 성과적으로 통합되였다는 사실은 개발자들의 과제가 완성되였다는것을 의미하지 않는다. SQA그룹은 여전히 그 제품이 성공적이라는것을 확정하기 위한 여러가지 시험과제를 수행하여야 한다. 쏘프트웨어에는 두가지 기본류형 즉 상업적인 규격쏘프트웨어(COTS)(2.1)와 주문쏘프트웨어(*custom software*)가 있다. COTS쏘프트웨어를 개발할 때의 목적은 그 제품이 될수록 많은 구매자들에게 팔린다는것을 보증하는것이다. 그러므로 COTS쏘프트웨어의 모든 모듈들이 성과적으로 통합되였을 때 제품은 제품시험을 위하여 SQA그룹에 넘겨 진다. COTS제품시험의 목적은 제품전체로서 오유가 없다는것을 보증하는것이다. 제품시험이 완결될 때 그 제품은 2.6.1에서 서술한 바와 같이 알파시험과 베타시험을 거치게 된다. 즉 SQA팀이 스쳐 지나 간 제품의 잔류

오유와 관련한 의견을 받을 목적으로 제품의 초기판본을 선발된 장래의 제품구입자들에게 발송한다.

한편 주문쏘프트웨어는 이와는 약간 다른 제품시험을 거치게 된다. SQA그룹은 여러가지 시험과제를 수행하여 주문쏘프트웨어개발팀이 최종적으로 극복하여야 할 장애물로 되는 인수시험에서 그 제품이 실패하지 않는다는것을 확인한다. 인수시험을 거치게 될어떤 제품에서의 오유는 거의 언제나 개발기업체의 경영능력에 대한 나쁜 반영을 초래한다. 의뢰자는 개발자들이 무능력하다고 단정할수 있으며 이로부터 의뢰자는 거의 확정적으로 이 개발자들을 다시는 채용하지 않기 위하여 할수 있는 모든것을 다하게 된다. 더 나쁘게 의뢰자는 개발자들이 불성실하고 계약을 빨리 결속하여 될수록 빨리 보수를 받으려고 고의적으로 비표준쏘프트웨어를 넘겨 주었다고 믿을수도 있다. 만일 의뢰자가 진정 이렇게 생각하고 다른 잠정적사용자들에게 말하게 되면 개발자들은 하나의 심각한 사회적비난에 직면하게 된다.

SQA그룹이야말로 제품이 인수시험을 성공적으로 거친다는것을 확증할 의무가 있다. 성공적인 인수시험을 담보하기 위하여 SQA그룹은 앞으로 하게 될 인수시험에 매우 근사하다고 믿어 지는 시험방법을 리용하여 제품을 시험하여야 한다.

1. 검은통시험실례들은 제품전체에 대하여 실행되여야 한다. 지금까지 시험실례들은 매 모듈 또는 객체들이 개별적으로 자기의 명세서들을 만족시킨다는것을 담보하면서 한 모듈씩 또는 한 객체씩 구성되였다.
2. 제품전체로서의 로바스트성이 시험되여야 한다. 게다가 개별적모듈들과 객체들의 로바스트성은 통합과정에 시험되였기때문에 이제는 제품전반의 로바스트성이 론점으로 되는데 바로 이런 견지에서 시험들을 구성하고 실행하여야 한다. 이밖에 제품은 강도시험(*stress testing*)을 받아야 한다. 즉 모든 말단들이 같은 시각에 등록가입하려고 한다든가 또한 손님들이 모든 자동출납기들을 동시에 리용한다든가와 같은 최대부하조건에서 제품을 조작할 때 그것이 정확하게 동작한다는것을 확인하여야 한다. 제품은 또한 용량시험(*volume testing*)을 받아야 한다. 실례로 제품이 큰 입력파일을 처리할수 있다는것을 확인하여야 한다.
3. SQA그룹은 제품이 모든 제약들을 만족시킨다는것을 검열하여야 한다. 실례로 명세서가 제품이 완전부하조건하에서 작업할 때 95%의 질문에 대한 응답시간이 3s 이하여야 한다는것을 서술하고 있다면 실제로 그렇게 된다는것을 검증하는것은 SQA그룹의 책임으로 된다. 의뢰자가 인수시험기간에 제약들을 검열한다는것은 자명하다. 만일 제품이 어떤 중요한 제약을 만족시키지 못하게 되면 개발기업체는 상당한 신용을 잃게 될것이다. 이와 류사하게 기억에 관한 제약과 보안과 관련한 제약들이 검열되여야 한다.
4. SQA그룹은 모든 문서들이 코드와 함께 의뢰자에게 넘어 간다는것을 검토하여야 한다. 또한 SQA그룹은 문서가 SPMP에 규정된 기준에 맞는다는것을 검열하여야 한다. 이밖에 문서는 제품에 대치하여 시험되여야 한다. 실례로 SQA그룹은 사용지도서가 그 제품을 리용하기 위한 정확한 방법들을 실지로 반영하고 있으며 제품의 기능들이 사용지도서에 명시된것과 같다는것을 결정하여야 한다.

객체지향쏘프트웨어의 제품시험에서 대본들(12.4)이 쓸모 있을수 있다. 제품의 거동이 명백히 규정된것과 같다는것을 보증하기 위하여 제품은 전체로서 매개의 대본에 기초하여 자세히 검토된다. 대본검열은 개별적인 모듈과 객체들의 검토는 물론 설계검토과정에도 유용하다. 그러나 대부분의 대본들 특히는 결정적인 대본들은 모듈과 객체의 경계에 놓이므로 제품시험기간에만 대본검열을 하는것이 가장 완전한 능력을 발휘할수 있게 한다.

일단 SQA그룹이 관리자측에게 제품이 인수시험자들이 제시하는 그 어떤 문제들도 처리할수 있다는것을 보증하면 그 제품(즉 코드와 모든 명세서들)은 인수시험를 위하여 의뢰자측에 넘겨 진다.

15. 5. 인수시험

인수시험의 목적은 의뢰자가 그 제품이 개발자들이 주장하는것처럼 명세서들을 실지로 만족시키는가를 결정하도록 하는것이다. 인수시험은 의뢰자측 또는 의뢰자측 대표자들의 면전에 있는 SQA그룹 또는 이 목적을 위하여 의뢰자들이 채용한 독립적인 SQA그룹에 의하여 진행된다. 인수시험은 자연히 정확성시험을 포함하지만 이밖에도 성능과 로바스트성을 시험하는것이 필요하다. 인수시험의 네가지 중요한 요소는 정확성, 로바스트성, 성능 및 문서의 시험인데 이것들은 명백히 개발자들이 제품시험기간에 진행하는 시험들이다. 제품시험은 인수시험을 위한 하나의 종합적인 시연회이기때문에 여기에 놀랄 필요는 없다.

인수시험의 한가지 중요한 측면은 그것이 시험자료가 아니라 실지자료에 기초하여 수행되여야 한다는것이다. 시험실례들이 아무리 잘 작성되고 자연스럽다고 하여도 그것들은 인공적인것이다. 더 중요하게는 시험자료가 대응하는 실지자료의 어떤 진실한 반영으로 되여야 하지만 실천적으로는 늘 그렇게 되지는 않는다. 실례로 실지자료를 특징 짓는데 책임이 있는 명세작성팀의 성원들은 이 과제를 부정확하게 수행할수도 있다. 반대로 비록 자료는 정확하게 명시된다고 하여도 그 자료명세서를 리용하는 SQA그룹성원이 그것을 오해할수도 있다. 그 결과로 생성된 시험실례들은 실지자료에 대한 진실한 반영으로 되지 못하며 부적당하게 시험된 어떤 제품을 초래하게 된다.

이러한 리유들로 하여 인수시험은 실지자료에 대하여 진행되여야 한다. 더우기 개발팀은 제품시험이 인수시험의 모든 측면들을 중복할것이라는것을 담보하려고 노력하기때문에 제품시험도 역시 될수록 실지자료에 대하여 수행되여야 한다.

어떤 새 제품이 현존제품을 대신하게 될 때 명세서에는 거의 언제나 그 새 제품이 현존제품과 병렬로 실행되도록 설치되여야 한다는 취지의 조항을 포함한다. 그 리유는 새 제품이 어떤 방식으로든 실패할수 있는 실제적가능성이 존재하기때문이다. 현존 제품은 정확하게 동작하지만 일부 측면에서 부적당하다. 만일 현존 제품이 부정확하게 동작하는 어떤 새 제품으로 교체된다면 의뢰자는 난관에 봉착하게 된다. 그러므로 의뢰자가 새 제품이 현존하는 제품의 기능들은 대신할수 있다는것을 인정할 때까지는 두 제품이 병렬로 실행되여야 한다. 병렬실행이 성공하면 인수시험을 결속하고 현존제품을 페기할수 있다.

제품이 인수시험을 거쳤을 때 개발자의 과제가 완수된다. 이제 그 제품에 가해 지는 임의의 변경들은 유지정비를 위하여 진행될 따름이다.

15. 6. 실현 및 통합단계에서의 CASE도구

실현을 지원하기 위한 CASE도구들은 제5장에서 약간 자세히 서술하였다. 실현 및 통합단계의 통합요소들을 위하여 판본조종도구, 구축도구, 구성관리도구들이 요구된다(5장). 그 리유는 시험되고 있는 모듈들은 오유들이 발견되고 정정됨에 따라서 끊임없이 변경되며 이러한 CASE도구들은 매 모듈의 적당한 판본들이 콤파일되고 링크된다는것을 담보하는데서 필수적인것으로 되기때문이다. 앞에서도 언급한바와 같이 세가지 중요한 UNIX판본조종도구들은 ***sccs***(*source code control system*; 원천코드조종체계) [Rochkind, 1975], ***rcs***(*revision control system*; 개정판조종체계) [Tichy, 1985], ***cvs***(*concurrent versions system*; 병행판본체계)[Loukides and Oram, 1997]이다. 시장에서 구입할수 있는 구성조종작업프로그람으로서는 PVCS와 SourceSafe를 들수 있다.

지금까지 매장마다 그 단계에 특정한 CASE도구들과 작업대들을 서술하였다. 이제는 개발공정의 모든 단계가 서술되였으며 제품전체를 위한 CASE도구를 고찰할 차례이다.

15. 7. 완전한 쏘프트웨어공정에서의 CASE도구

CASE내에는 자연스러운 진보과정이 존재한다. 5.6에서 설명한것처럼 가장 단순한 CASE수단은 직결식대면부검사기(*online interface checker*) 또는 구축도구(*build tool*)와 같은 단일도구이다. 다음으로 도구들이 결합되여 구성조종 또는 코드작성과 같은 쏘프트웨어개발공정에서의 하나 또는 두개의 활동을 지원하는 작업대(*workbench*)가 만들어 졌다. 그러나 전체에 대하여서는 말할것도 없고 그것이 리용될수 있는 쏘프트웨어개발공정의 제한된 범위에서조차 경영진에게 정보를 제공하지 못할수 있다. 결국 개발환경(*environment*)이 비록 전부는 아니라 해도 대부분의 개발공정에 대하여 콤퓨터에 의한 지원을 제공하여 준다.

리상적으로 매 쏘프트웨어개발기업체들은 개발환경을 리용하여야 한다. 그러나 어떤 개발환경의 비용은 방대할수 있다. 즉 프로그람묶음 그자체뿐만아니라 그것을 실행할 하드웨어에 많은 비용이 든다. 보다 작은 개발기업체에는 작업대든가 또는 도구들의 모임이 적당할수 있다. 그러나 가능하다면 개발과 유지정비노력을 지원하기 위하여 어떤 통합화된 개발환경을 리용하여야 한다.

15. 8. 통합화된 개발환경

CASE와 관련한 범위내에서 단어《통합화된》의 가장 공통적인 의미는 사용자대면부통합(*user interface integration*)이라는 용어에 있다. 즉 그 환경안의 모든 도구들이 하나의 공통적인 사용자대면부를 공유한다. 이 리면에 있는 착상은 모든 도구들이 같은 시각적

외형을 가진다면 한개 도구를 다룰수 있는 사용자는 그 환경안의 다른 도구를 배우고 리용할 때 어려움이 적어야 한다는것이다. 이 착상은 마킨토쉬에서 성공적으로 실현되였는데 여기에서는 대다수의 응용들이 류사한 《보기와 느낌》을 가지고 있다. 비록 이것이 일반적인 의미라고 하여도 다른 류형의 통합도 있다.

용어 도구통합(*tool integration*)은 모든 도구들이 같은 자료형식을 통하여 통신한다는 것을 의미하고 있다. 실례로 UNIX Programmer's Workbench에서 UNIX pipe형식은 모든 자료들이 하나의 ASCII흐름형식으로 존재한다는것을 가정하고 있다. 결국 한 도구에서 나오는 출력을 다른 도구의 입력흐름에 지향시킴으로써 두개의 도구를 쉽게 결합한다.

공정통합(*Process integration*)은 하나의 특정한 쏘프트웨어개발공정을 지원하는 어떤 환경을 가리킨다. 이러한 부류의 환경들의 하나의 부분모임이 기법에 기초한 환경(*technique-based environment*)이다(다음의 《알고 싶은 문제》를 보시오.). 이러한 류형의 환경은 완전한 공정이 아니라 쏘프트웨어를 개발하는 어떤 특정한 기법만을 지원한다. 이 책에서 론의된 여러가지 기법들 즉 게인과 사르센의 구조화체계분석(11.3), 잭슨체계개발(13.5), 페트리망(11.7)들을 위한 환경들이 존재한다. 이러한 환경들의 대다수는 명세작성과 설계단계를 위한 도형적지원을 제공하며 자료사전을 병합한다. 일부 일치성검열들은 일반적으로 지원된다. 개발공정을 관리하기 위한 지원이 환경에 병합된다. Analyst/Designer와 Rhapsody를 비롯하여 이러한 류형의 환경들은 대부분 시장에서 구입할수 있다. Analyst/Designer는 상태도표(S*tatechart*)를 지원한다[Harel et al., 1990]. 객체지향방법들과 관련하여 Rose는 통합쏘프트웨어개발공정을 지원한다[Jacobson, Booch and Rumbaugh, 1999]. 이밖에 일부 낡은 환경들이 객체지향파라다임을 지원하도록 확장되였다. Software through Pictures가 이러한 류형의 한가지 실례이다. 거의 모든 객체지향환경은 UML을 지원하고 있다.

알고 싶은 문제

문헌들에서 기법에 기초한 환경(*technique-based environment*)을 보통 방법에 기초한 환경(*method-based programming*)이라고 부른다. 객체지향파라다임의 출현은 단어 방법(*method*)에 또 하나의 의미를 주었다(쏘프트웨어공학의 범위내에서). 그것의 원래의 의미는 기법(*technique*) 또는 방법(*approach*)이다. 이것이 이 단어가 방법에 기초한 환경(*method-based environment*)에 리용된 근거이다. 이 단어의 객체지향적의미는 7.7에서 설명한것처럼 어떤 객체 또는 클라스안의 작용이다. 유감스럽게도 때때로 어느 의미가 지적되는가는 문맥상 전혀 명백치 않다.

따라서 이 책에서는 객체지향파라다임의 범위내에서만 단어 방법(*method*)을 리용한다. 한편 용어 기법(*technique*) 또는 방법(*approach*)을 리용한다. 실례로 이것은 제11장에서 용어 형식적방법(*formal method*)이 나타나지 않는 리유이다. 그대신 이 책에서는 형식적기법(*formal technique*)을 리용한다. 이와 류사하게 이 장에서 용어 기법에 기초한 환경(*technique-based environment*)을 리용한다.

대부분의 기법에 기초한 환경들에서 쏘프트웨어개발에서의 수동적인 조작을 지원하고 형식화하는데 중점을 두고 있다. 즉 이러한 환경들은 사용자로 하여금 저자가 지적한

방식으로 기법들을 계단식으로 리용할수 있게 하며 한편 도형도구, 자료사전, 일치성검열들을 제공함으로써 사용자들을 지원한다. 이러한 콤퓨터화된 틀은 기법에 기초한 환경의 우점이며 여기서 사용자들은 특정한 기법들을 리용하여 그것을 정확하게 리용하게 된다. 그러나 그것은 약점으로 될수 있다. 개발기업체의 쏘프트웨어개발공정이 이러한 특정한 기법들을 병합하지 않는한 기법에 기초한 환경의 리용은 비생산성을 초래할수 있다.

15. 9. 업무응용을 위한 개발환경

한가지 중요한 부류의 개발환경이 업무지향제품을 구성하는데 리용된다. 여기서 중요한 점은 리용하기 쉬우며 여러가지 방식으로 실현된다는것이다. 특히 이 환경은 여러가지 표준화면들을 병합하고 있는데 이 화면들은 사용자지향 GUI발생기를 통하여 끝없이 변경될수 있다. 이와 같은 개발환경에서 한가지 보편적인 특성은 코드발생기이다. 그로부터 상세설계가 가장 낮은 준위의 제품추상화로 된다. 상세설계는 C, C++ 또는 Java와 같은 언어로 코드를 자동적으로 발생하는 코드발생기의 입력으로 된다. 자동적으로 발생된 이 코드가 콤파일된다. 즉 그우에서 객체지향환경들이 현재 UML을 지원하고 있다. 대다수의 기법에 기초한 환경에서 중점은 아무런 종류의 《프로그람작성》도 진행되지 않는다는것이다.

상세설계를 명시하기 위한 언어가 미래의 프로그람작성언어로 될것이다. 프로그람작성언어들의 추상준위는 1세대 및 2세대언어의 물리적기계준위로부터 3세대 및 4세대언어의 추상기계준위로 올라 간다. 현재 상세설계준위, 이식가능한 준위가 이러한 류형의 환경들의 추상준위로 된다. 14.2에서는 4세대언어를 리용하는 목적은 원천코드길이를 더 짧게, 개발을 더빨리 하며 유지정비를 더 쉽게 하는것이라는것을 설명하였다. 코드발생기의 리용은 프로그람작성자가 4GL을 위한 콤파일러나 해석기보다 코드발생기에서 보다 적은 세부를 진행하게 된다는 점에서 이러한 목적을 훨씬 더 달성할수 있게 한다. 결국 코드발생기를 지원하는 업무지향환경의 리용은 생산성을 증대시킬것으로 예상된다.

Foundation과 Bachman Product Set를 비롯한 이러한 류형의 여러가지 환경들은 현재 구입할수 있다. 업무지향CASE환경에 대한 시장규모를 고려하면 앞으로 이러한 류형의 개발환경들이 더 많이 개발될것 같다.

15. 10. 공용도구의 하부구조

정보기술연구를 위한 유럽전략계획(*European strategic Programme for Research in Information Technology*; ESPIT)은 CASE도구를 지원하기 위한 하나의 하부구조를 개발하였다. 그 이름과는 달리 이식가능한 공용도구환경(*portable common tool environment*; PCTE) [Thomas, 1989, and Long and Morris, 1993]은 개발환경이 아니다. 그대신 이것은 UNIX가 사용자제품에 필요한 연산체계봉사를 제공하는것과 같은 방식으로 CASE도구에 필요한 봉사를 제공하여 주는 하나의 하부구조이다(PCTE에서 단어《common》은 《public》 또는 《not copyrighted》의 의미로 쓰인다.).

PCTE는 광범히 응용되였다. 실례로 PCTE와 PCTE에 대한 C와 Ada대면부들은 1995년에 ISO/IEC규격13719로서 도입되였다. PCTE의 실현들은 Emeraude와 IBM의 실현들을 포함하고 있다.

앞으로 더욱더 많은 CASE도구들이 PCTE규격을 승인하게 될것이며 PCTE 그자체가 보다 광범한 종류의 콤퓨터상에서 실현될것으로 기대된다. PCTE를 승인하는 도구는 PCTE를 지원하는 임의의 콤퓨터상에서 실행될것이다. 따라서 이것은 CASE도구들의 많은 부분들이 광범히 응용되게 하여야 하며 이것은 보다 좋은 쏘프트웨어개발공정과 모든 질좋은 쏘프트웨어를 개발하는데로 이어 져야 한다.

15.11. 개발환경에서 제기될수 있는 문제

모든 제품들과 모든 개발기업체들에 리상적인 개발환경이란 없으며 하나이상의 프로그람작성언어가 《가장 좋다.》고 간주될수 있다. 이 개발환경들은 자기의 우점과 약점을 가지고 있으며 어떤 부적당한 개발환경을 선택하는것은 개발환경을 전혀 리용하지 않는것보다 못할수 있다. 실례로 15.8에서 설명한바와 같이 어떤 기법에 기초한 개발환경은 사실상 어떤 수동적인 과정을 자동화한다. 만일 어떤 개발기업체가 그 개발기업체전체에 또는 개발되고 있는 현재의 쏘프트웨어제품에 부적당한 기법을 강요하는 어떤 개발환경을 리용하도록 선택한다면 그러한 CASE환경의 리용은 비생산성을 초래하게 될것이다.

보다 나쁜 정황은 어떤 개발기업체가 5.10의 권고 즉 CASE환경의 리용은 개발기업체가 CMM준위 3에 도달할 때까지 피하여야 한다는것(2.11)을 무시하고 선택하는 경우에 발생한다. 물론 매개의 개발기업체는 CASE도구를 리용하여야 하지만 일반적으로 작업대를 리용할 때 손해가 적다. 그러나 개발환경은 그것을 리용하는 어떤 개발기업체들에게 자동화된 쏘프트웨어개발공정을 강요한다. 만일 좋은 개발공정을 리용한다면 즉 개발기업체가 3준위 또는 그보다 높은 준위에 있다면 그 개발환경의 리용은 공정을 자동화함으로써 쏘프트웨어생산의 모든 측면을 도와 주게 될것이다. 그러나 만일 개발기업체가 위험구동준위 1이든가 지어 2준위에 있다면 이와 같은 공정은 그 어디에도 없다. 이러한 실제하지 않는 공정을 지동화하는것 즉 CASE개발환경(CASE도구나 CASE작업대에 대립하는것으로서)의 도입은 혼란만을 초래할수 있다.

15.12. 실현 및 통합단계에서의 척도

코드행과 맥케브의 주기적복잡도를 비롯하여 실현 및 통합단계에서의 여러가지 각이한 복잡도에 관한 척도들이 14.8.2에서 론의되였다. 시험의 관점으로부터 그와 련관된 척도들에는 시험실례의 총 개수와 오유를 초래하는 시험실례의 개수가 포함된다. 일반적인 오유통계를 코드시험을 위하여 유지하여야 한다. 오유의 총 개수도 중요하다. 왜냐하면 어떤 모듈이나 객체에서 발견된 오유의 개수가 사전에 결정한 어떤 최대값을 초과하면 14.14에서 론의한바와 같이 그 모듈이나 객체는 다시 설계하고 다시 코드작성해야 하기때문이다. 이밖에 발견된 오유의 류형과 관련하여 세부적통계들을 보관해 둘 필요가 있

다. 전형적인 오유류형들에는 설계에 대한 오해, 초기화의 결여, 변수리용에서의 불일치성이 포함된다. 오유자료들은 시험목록들에 병합되여 장래의 제품에 대한 코드시험기간에 리용되게 된다.

15.13. 항공음식전문회사 실례연구: 실현 및 통합단계

항공음식전문회사제품에 대한 C++와 Java에 의한 완전한 실현은 www.mhhe.com/engcs/compsci/schach로부터 내려 받을수 있다. 이 실현은 상세설계(부록 8과 부록 9)의 코드에로의 직접번역으로 간주할수 있다. 즉 설계자들은 프로그람작성팀에게 순전히 C++ 또는 Java로 실현하도록 요구하는 심사숙고한 한가지 설계를 제출하였다. 프로그람작성자들은 여기서 유지정비를 하는 프로그람작성자들을 돕기 위하여 설명문을 포함하였다. 이 실현은 문제 14.21부터 14.25까지의 흰통시험실례들은 물론 부록 10의 검은통시험실례들에 대하여 시험되였다.

15.14. 실현 및 통합단계에서의 난관

순수한 기술적관점으로부터 보면 실현 및 통합단계는 상대적으로 간단하다. 만일 요구사항확정, 명세작성(분석), 설계단계들이 만족스럽게 수행되면 실현 및 통합의 과제들은 유능한 프로그람작성자들에게 적은 문제점을 제출하여야 한다. 그러나 실현 및 통합단계의 관리는 매우 중요하다. 실현 및 통합단계에서의 난관은 이 분야에서 발생한다.

적당한 CASE도구들의 리용(15.11), 일단 명세서가 의뢰자들에 의하여 수표된 시험계획(9.8.6), 설계에 대한 변경이 모든 련관된 사람들에게 전달된다는것을 담보하는것(15.15), 시험을 중지하고 제품을 의뢰자에게 배포할 시기를 결정하는것(6.1.2)들이 제품의 성패와 관련된 대표적인 론점들이다.

쏘프트웨어개발공정의 효과적인 관리는 효과적인 쏘프트웨어생산에서 본질적인 문제이다. 즉 이것은 능력성숙도모형(2.11)과 같이 쏘프트웨어개발공정을 개선하기 위한 노력을 유발시키게 된다. 그러나 이 절의 첫 부분에서 서술한바와 같이 관리는 실현 및 통합단계에서 특별히 중요한 문제로 제기된다.

요 약

실현 및 통합활동들은 병렬로 진행되여야 한다(15.1). 하강식, 상승식, 쌘드위치식실현 및 통합이 서술되고 서로 비교된다(15.1.1부터 15.1.3까지). 객체지향제품의 실현 및 통합은 15.1.4에서 론의된다. 제품시험(15.4)과 인수시험(15.5)을 비롯한 각이한 류형의 시험이 실현 및 통합단계에서 수행되여야 한다(15.2). 도형사용자대면부의 통합시험에 의하여 특수한 문제들이 제기될수 있다(15.3). 15.6에서는 실현 및 통합단계에서의 CASE도구들에 대하여 론의한다. 완전한 개발공정에서의 CASE도구들은 15.7절에서 론의된다. 통합화된

CASE도구들에 대한 문제는 15.7에서 론의된다. 15.10은 공용도구의 하부구조에 대하여 주목을 돌리고 있다. 다음으로 개발환경과 관련된 잠재적문제들이 론의되며 (15.11) 실현 및 통합단계에서의 척도들이 론의된다(15.12). 이 장은 항공음식전문회사제품의 실례연구에 대한 실현 및 통합(15.13)과 실현 및 통합단계에서의 난관에 대한 론의(15.14)로써 결속된다.

보 충

통합시험를 위한 시험자료의 선택은 문헌 [Harrold and Soffa, 1991]에 제시되였다. 문헌 [Munoz, 1988]은 제품시험에 대한 한가지 방법을 서술하고 있다. 어떤 제품시험을 계속하는데 얼마동안 높은 비용효률을 유지하겠는가를 파악하는것이 중요하다. 이 문제에 대한 해결방안은 문헌 [Brettschneider, 1989, and Sherer, 1991]에 서술되였다. 대규모제품의 시험은 문헌 [House and Newman, 1989]에 서술되였다.

Software through Pictures [Wasserman and Pircher, 1987]을 비롯한 여러가지 기법에 기초한 개발환경들이 존재한다. 통합화된 CASE도구들에 대한 정보는 *IEEE Software* 1992년 3월호에서 특히 문헌 [Brown and McDermid, 1992, and Chen and Norman, 1992]에서 찾아볼수 있다.

2년 또는 3년마다 ACM SIGSOFT와 SIGPLAN은 Symposium on Practical Software Development Environments를 발행한다. 이 회보는 도구모임과 개발환경에 대한 넓은 범위의 정보를 제공하여 준다. 또한 the annual International Workshops on Computer Aided Software Engineering에 대한 회보도 쓸모 있다. 개발환경에 대한 그이상의 론문들은 *IEEE Software* 1992년 5월호를 비롯한 각이한 잡지들의 전문주제들에서 찾아 볼수 있다.

객체지향파라다임과 관련하여 문헌 [Jorgensen and Erickson, 1994]는 객체지향쏘프트웨어에 대한 통합시험을 서술하고 있다. *Communications of the ACM* 1994년 9월호는 객체지향쏘프트웨어의 시험에 대한 여러개의 론문을 포함하고 있다.

PCTE에 대한 개요는 문헌 [Thomas, 1989]에서 찾아 볼수 있다. 문헌 [Long and Morris, 1993]은 PCTE에 대한 많은 정보자원을 포함하고 있다.

문 제

15.1. 론리모듈과 조작모듈의 차이를 설명하시오.

15.2. 방어프로그람작성은 좋은 쏘프트웨어공학실천으로 된다. 동시에 그것은 조작모듈이 재리용목적에 충분하도록 철저히 시험되는것을 가로 막는다. 이러한 명백한 모순은 어떻게 해결할수 있는가?

15.3. 제품시험과 인수시험의 류사성은 무엇인가? 기본차이는 무엇인가?

15.4. SQA그룹이 실천 및 통합단계에서 노는 역할은 무엇인가?

15.5. 당신은 1인쏘프트웨어회사의 주인이고 유일한 직원이다. 당신은 경쟁을 위하여 CASE도구를 사야 한다고 결심한다. 그러므로 당신은 15,000딸라의 은행대부를 요청한다.

은행경영자는 당신에게 왜 CASE도구를 필요로 하는가를 일반용어로 설명하는 1페지를 넘지 않는(더 짧으면 좋다.) 설명문을 요구한다. 그 설명문을 작성하시오.

15.6. Ye Olde Fashioned Software Corporation회사의 쏘프트웨어개발부의 새로 임명된 부책임자는 회사가 쏘프트웨어를 개발하는 방식을 변경하는데서 방조를 받을 목적으로 당신을 채용하였다. 회사에는 650명의 직원이 있으며 그들은 모두 아무런 CASE도구의 도움도 없이 COBOL코드를 작성하고 있다. 회사가 어떤 종류의 CASE도구를 구입하여야 하는가를 설명하는 부책임자에게 내는 비망록을 작성하시오. 당신의 선택을 주의깊게 증명하시오.

15.7. 당신과 한 동료가 Personal Computer Software Programs Are Us에 착수하여 개인콤퓨터상에서 개인콤퓨터용쏘프트웨어를 개발하기로 결심하였다. 그리고 먼 친척이 죽으면서 당신이 그 돈을 업무지향개발환경과 그것을 실행하는데 필요한 하드웨어를 구입하고 그 개발환경을 적어도 5년간 유지하는데 소비한다는 조건하에서 당신에게 백만딸라의 자금을 넘겨 주었다. 당신은 무엇을 해야 하며 왜 그렇게 해야 하는가?

15.8. 당신은 어떤 우수한 소규모문예단과대학의 콤퓨터과학교수이다. 콤퓨터과학과정안을 위한 과목함량이 35개의 개인콤퓨터를 망라한 망우에서 실현된다. 학장이 어떤 종류의 싸이트리용허가가 허락되지 않는 한 매 CASE도구의 복제본을 구입해야 한다는것을 념두에 두면서 당신에게 제한된 쏘프트웨어예산을 리용하여 CASE도구들을 사겠는가 말겠는가를 묻는다. 당신은 이때 무엇을 권고하겠는가?

15.9. 당신은 어떤 중요한 도시의 시장으로 금방 선거되였다. 도시를 위한 쏘프트웨어를 개발하는데 그 어떤 CASE도구들도 리용되지 않고 있다는것을 당신은 발견하였다. 당신은 무엇을 하겠는가?

15.10. (과정안상 목표) 브로드랜즈지역 아동병원제품(부록 1)을 실현하고 통합하시오. 교원이 지정하는 프로그람작성언어를 리용하시오. 교원이 당신에게 web에 기초한 사용자대면부든가 도형사용자대면부든가 본문에 기초한 사용자대면부를 구성한다고 말할것이다. 당신이 작성한 코드를 시험하기 위하여 문제 14.20에서 개발한 검은통시험실례들을 리용하는것을 기억하시오.

15.11. (실례연구) 13.13의 상세설계로 출발하여 C++ 또는 Java가 아닌 객체지향언어로 항공음식전문회사제품의 실례연구를 코드작성하시오.

15.12. (실례연구) C++특성이 없는 순수한 C로서 항공음식전문회사제품의 실례연구(15.13)를 재작성하시오. 비록 C가 계승을 지원하지 않을지라도 교갑화와 정보은페와 같은 객체에 기초한 개념들이 쉽게 달성될수 있다. 다형성과 동적결합을 어떻게 실현하겠는가?

15.13. (실례연구) 15.13의 실현에 대한 코드문서가 어느 정도로 부적당한가? 임의의 필요한 보충을 하시오.

15.14. (쏘프트웨어공학독본) 교원은 문헌 [Jorgensen and Erickson, 1994]의 복제본을 배포할것이다. 객체지향통합과 고전적통합의 기본차이는 무엇인가?

참고문헌

[Brettschneider, 1989] R. BRETTSCHNEIDER, "Is Your Software Ready for Release?" *IEEE Software* **6** (July 1989), pp. 100, 102, and 108.

[Brown and McDermid, 1992] A. W. BROWN AND J. A. MCDERMID, "Learning from IPSE's Mistakes," *IEEE Software* **9** (March 1992), pp. 23–29.

[Chen and Norman, 1992] M. CHEN AND R. J. NORMAN, "A Framework for Integrated CASE," *IEEE Software* **8** (March 1992), pp. 18–22.

[Harel et al., 1990] D. HAREL, H. LACHOVER, A. NAAMAD, A. PNUELI, M. POLITI, R. SHERMAN, A. SHTULL-TRAURING, AND M. TRAKHTENBROT, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering* **16** (April 1990), pp. 403–14.

[Harrold and Soffa, 1991] M. J. HARROLD AND M. L. SOFFA, "Selecting and Using Data for Integration Testing," *IEEE Software* **8** (1991), pp. 58–65.

[House and Newman, 1989] D. E. HOUSE AND W. F. NEWMAN, "Testing Large Software Products," *ACM SIGSOFT Software Engineering Notes* **14** (April 1989), pp. 71–78.

[Jacobson, Booch, and Rumbaugh, 1999] I. JACOBSON, G. BOOCH, AND J. RUMBAUGH, *The Unified Software Development Process,* Addison-Wesley Longman, Reading, MA, 1999.

[Jorgensen and Erickson, 1994] P. C. JORGENSEN AND C. ERICKSON, "Object-Oriented Integration Testing," *Communications of the ACM* **37** (September 1994), pp. 30–38.

[Long and Morris, 1993] F. LONG AND E. MORRIS, "An Overview of PCTE: A Basis for a Portable Common Tool Environment," Technical Report CMU/SEI–93–TR–1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, January 1993.

[Loukides and Oram, 1997] M. K. LOUKIDES AND A. ORAM, *Programming with GNU Software,* O'Reilly and Associates, Sebastopol, CA, 1997.

[Munoz, 1988] C. U. MUNOZ, "An Approach to Software Product Testing," *IEEE Transactions on Software Engineering* **14** (November 1988), pp. 1589–96.

[Myers, 1979] G. J. MYERS, *The Art of Software Testing,* John Wiley and Sons, New York, 1979.

[Rochkind, 1975] M. J. ROCHKIND, "The Source Code Control System," *IEEE Transactions on Software Engineering* **SE-1** (October 1975), pp. 255–65.

[Sherer, 1991] S. A. SHERER, "A Cost-Effective Approach to Testing," IEEE Software **8** *(March 1991), pp. 34–40.*

[Thomas, 1989] I. THOMAS, "PCTE Interfaces: Supporting Tools in Software Engineering Environments," *IEEE Software* **6** (November 1989), pp. 15–23.

[Tichy, 1985] W. F. TICHY, "RCS—A System for Version Control," *Software—Practice and Experience* **15** (July 1985), pp. 637–54.

[Wasserman and Pircher, 1987] A. I. WASSERMAN AND P. A. PIRCHER, "A Graphical, Extensible Integrated Environment for Software Development," Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, *ACM SIGPLAN Notices* **22** (January 1987), pp. 131–42.

[Yourdon, 1989] E. YOURDON, *Modern Structured Analysis,* Yourdon Press, Englewood Cliffs, NJ, 1989.

제1 6장. 유지정비단계

제품이 일단 인수시험을 거치게 되면 그것은 의뢰자에게 넘어 간다. 제품은 설치되여 그것을 만들 목적에 리용되게 된다. 그러나 모든 유용한 제품들은 거의 확정적으로 유지정비단계에서 유지정비를 거치게 되며 오유를 수정하거나(교정유지정비) 또는 제품의 기능을 확장(확대)하게 된다.

제품은 원천코드만으로 구성되지 않기때문에 제품이 의뢰자에게 넘어 간 다음에 그 제품의 문서, 지도서나 기타 요소들에 가해 지는 모든 변경들은 다 유지정비의 실례로 된다. 일부 콤퓨터과학자들은 제품이 시간에 따라 진화된다는것을 지적하기 위하여 유지정비라는 용어보다 진화(*evolution*)라는 용어를 더 즐겨 쓴다. 사실 일부 사람들은 전체 쏘프트웨어생명주기의 시작부터 끝까지를 하나의 진화과정으로서 간주한다.

이 책의 한가지 기본주제는 유지정비의 사활적인 중요성에 대한 문제이다. 그러므로 이 장이 상대적으로 짧은데 대하여 놀랄수도 있다. 그 리유는 유지정비성이 제품개발공정의 첫 시기부터 제품에 구현되여야 하며 개발공정의 그 어느 시기에도 양보하지 말아야 하는 문제이기때문이다. 그렇기때문에 실제적인 의미로 보면 선행한 모든 장들에서 유지정비라는 주제에 품을 들여 왔다. 이 장에서는 유지정비성이 유지정비단계 그자체에서 양보되지 않는다는것을 어떻게 담보하겠는가에 대하여 서술한다.

16. 1. 유지정비의 필요성

제품에 대하여 변경을 가하여야 할 세가지 기본 리유가 있다.

- 첫번째 리유는 오유들 즉 명세서오유, 설계오유, 코드작성오유, 문서작성오유 또는 기타 류형의 오유들을 정정하는것이다. 이것을 교정유지정비(*corrective maintenance*)라고 부른다. 놀랍게도 65개 개발기업체들에 대한 하나의 연구자료는 유지정비프로그람작성자들이 교정유지정비에 자기들에게 부여된 시간의 17.5%만을 소비한다는것을 보여 주었다[Lientz, Swanson, and Tompkins, 1978]. 이것은 그림 16-1의 파라다임도표에서 보여 주고 있다.

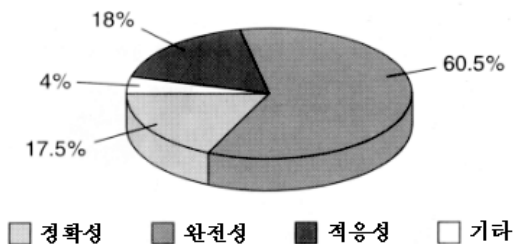



그림 16-1. 매 류형의 유지정비에 드는 시간의 퍼센트

○ 유지정비프로그람작성자들은 대부분의 소여시간 즉 60.5%의 시간은 두번째 류형의 유지정비 즉 완전유지정비(*perfective maintenance*)에 소비하였다. 여기에서는 제품의 효률성을 개선할 목적으로 코드에 대한 변경이 진행되였다. 실례로 의뢰자는 보충적인 기능을 추가하거나 제품이 더빨리 실행되도록 그 제품이 변경되기를 바랄수 있다. 어떤 제품의 유지정비성을 개선하는것은 완전유지정비의 또 한가지 실례로 된다.

○ 제품을 변경시켜야 할 세번째 리유는 적응유지정비(*adaptive maintenance*)인데 이때에는 제품이 동작하는 환경에서의 변화에 그 제품이 적응하도록 변경이 가해진다. 실례로 어떤 제품은 새로운 콤파일이나 조작체계, 하드웨어에 이식되게 된다면 거의 확정적으로 갱신되여야 한다. 세금규약에서의 변경이 진행될 때마다 세금신고서를 준비하는 어떤 제품도 그에 따라 변경되여야 한다. 어느 한 우편봉사업체가 1981년 9수자 ZIP코드를 도입하였을 때 5개 수자 ZIP코드만 허용하였던 제품들이 변경되지 않으면 안되였다. 적응유지정비는 의뢰자가 요청하지 않지만 그대신 의뢰자에게 의무적으로 강요된다. 연구결과는 쏘프트웨어유지정비의 18%가 사실상 적응유지정비이라는것을 보여 주었다.

유지정비시간의 나머지 4%는 세가지 부류에 속하지 않는 다른 류형의 유지정비에 돌려 졌다.

16. 2. 유지정비프로그람작성자들의 요구

쏘프트웨어생명주기에서 다른 단계들보다 유지정비에 더 많은 시간이 소비된다. 그림 12-2에서 보여 준바와 같이 사실상 제품개발에 드는 총 비용에서 평균적으로 적어도 67%가 유지정비에 소비된다. 그러나 많은 개발기업체들은 현재까지도 보다 우수하거나 보다 경험이 많은 프로그람작성자들에게는 제품개발을 진행하는 《매력적인》업무를 맡기면서 유지정비과제는 초학자들과 능력이 부족한 프로그람작성자들에게 맡기고 있다.

사실상 유지정비는 쏘프트웨어생산의 모든 측면가운데서 가장 어려운 과제이다. 한가지 중요한 리유는 유지정비가 쏘프트웨어개발공정의 기타 모든 단계들의 측면을 포괄하고 있기때문이다. 어떤 오유보고서가 유지정비프로그람작성자에게 넘어 왔을 때 무슨 일이 발생하겠는가를 고찰하여 보자. 사용자의 견지에서 제품이 사용자안내서에 명시된대로 동작하지 않는 경우에 오유보고서가 제출된다. 여러가지 가능한 원인들이 존재한다. 첫째로, 전혀 아무것도 틀리지 않았을수도 있다. 즉 아마도 사용자가 사용자안내서를 잘못 리해하였거나 제품을 부정확하게 리용하고 있을수 있다. 이와 같이 만약 오유가 제품에 있다면 단순히 사용자안내서가 잘못 작성되고 코드 그자체에는 아무런 영향이 없을수도 있다. 그럼에도 불구하고 일반적으로 오유는 코드에 있다. 그러나 유지정비프로그람작성자는 그 어떤 변경을 가하기전에 사용자가 제출한 오유문서, 원천코드 그리고 그밖의 다른것들을 리용하여 어디에 오유가 있는가를 정확히 결정해야 한다. 그러므로 유지정비프로그람작성자에게는 평균수준보다 훨씬 높은 오유수정능력이 필요하다. 왜냐하면 오유는 제품의 그 어디에나 존재할수 있기때문이다. 그리고 오유의 근본원인이 아직까지는

존재하지 않는 명세서나 설계문서들에 있을수도 있다.

유지정비프로그람작성자가 오유를 찾아 내고 제품의 다른 곳에서 실수하여 다른 오유 즉 회귀오유(*regression fault*)를 범하지 않고 그 오유를 수정한다고 가정하자. 만일 회귀오유가 최소로 되게 되면 제품전체와 개별적제품들에 대한 세부문서작성을 리용할수 있어야 한다. 그러나 쏘프트웨어전문가들은 모든 종류의 문서작업, 특히 문서작성을 하기 싫어 한다는것은 누구나 다 아는 사실이며 또한 문서작성이 불충분하거나 오유가 있으며 완전히 틀릴수도 있다는것은 매우 일반적인 사실이다. 이러한 경우에 유지정비프로그람작성자는 리용할수 있는 유일하게 정당한 문서형태인 원천코드로부터 어떤 회귀오유의 인입을 피하기 위하여 필요한 모든 정보들을 추출해 내야 한다.

가능한 오유를 결정하고 그것을 정정하려고 시도한 다음 유지정비프로그람작성자는 변경된 제품이 정확히 동작하며 그 어떤 회귀오유도 인입되지 않았다는것을 시험하여야 한다. 변경 그자체를 검열하기 위하여 유지정비프로그람작성자는 특수한 시험실례들을 만들어야 한다. 한편 회귀오유에 대한 검열은 회귀시험을 수행하기 위하여 정확하게 저장한 시험자료들의 모임을 리용하여 수행된다(2.7.1). 그다음 변경을 검열하기 위하여 구성한 시험실례들이 저장된 시험실례들에 추가되여 변경된 제품에 대한 이후의 회귀시험에 리용되게 된다. 이밖에 만일 오유를 정정하기 위하여 명세서나 설계에 대한 변경이 진행된다면 이러한 변경들도 역시 검열되여야 한다. 그러므로 시험에서는 전문지식이 유지정비를 위한 추가적인 전제조건으로 된다. 결국 유지정비프로그람작성자가 모든 변경내용과 문서를 작성하는것이 본질적인 문제로 된다. 이 과제를 수행하기 위하여 유지정비프로그람작성자는 우선 오유의 존재여부를 결정하는 최고수준의 진단전문가로 되여야 하며 만일 그렇게 된다면 그것을 수정하는 전문기술자가 되여야 한다.

그러나 기본유지정비과제들은 적응유지정비와 완전유지정비이다. 이 과제를 수행하기 위하여 유지정비프로그람작성자는 현존제품을 출발점으로 하여 요구사항확정 , 명세작성, 설계, 실현 및 통합단계들을 거쳐야 한다. 일부 류형의 변경들에 대하여서는 추가적인 모듈들을 설계하고 실현하여야 한다. 다른 경우들에 현존모듈들의 설계와 실현에 대한 변경이 요구된다. 결국 명세서들은 흔히 명세작성전문가가 작성하고 설계는 설계전문가가, 코드는 프로그람작성전문가가 작성하는 반면에 유지정비프로그람작성자는 이 세개의 령역에서 모두 전문가로 되여야 한다. 완전유지정비와 적응유지정비는 교정유지정비와 마찬가지로 적당한 문서작성의 결과로 인하여 거꾸로 영향을 받았다. 더우기 교정유지정비에서는 마찬가지로 완전유지정비와 적응유지정비를 위하여 적당한 시험실례들을 설계하고 훌륭한 명세서도 작성할수 있는 능력이 요구된다. 그러므로 최고급콤퓨터전문가가 진행공정을 관리하지 않는한 에 있어서 그 어떤 형식의 유지정비도 경험이 어린 프로그람작성자의 과제로 될수 없다.

선행한 론의로부터 유지정비전문가는 쏘프트웨어전문가가 가질수 있는 모든 기술적 기능들을 거의 다 소유하여야 한다는것이 명백하다. 그러나 그가 무엇을 보상 받겠는가?

1. 유지정비는 그 어떤 방식으로도 감사를 받지 못할 과제이다. 유지정비성원들은 불만족해 하는 사용자들을 취급한다. 만일 사용자가 제품에 만족해 한다면 유지정비는 필요 없을것이다.
2. 사용자가 제기하는 문제점들은 흔히 유지정비성원이 아니라 제품을 개발한 개별

적사람들에 의하여 초래된것이다.

3. 코드 그자체가 잘못 작성되여 유지정비성원의 실패에 추가될수 있다.
4. 유지정비는 대부분의 프로그람작성자들로부터 차요시되는데 그들은 제품개발은 매혹적인 업무로 간주하며 유지정비는 중간급프로그람작성자들이나 무능력한 프로그람작성자들에게만 적합한 단조로운 일로 생각한다.

유지정비는 판매후의 봉사에 비유할수 있다. 제품이 의뢰자에게 넘겨 졌다고 하자. 그러나 그 제품이 정확하게 동작하지 않거나 의뢰자가 일상적으로 요구하는 모든 동작을 하지 못하거나 제품이 만들어 질 때의 환경이 어떤 방식으로든지 변화된것으로 인하여 의뢰자는 현재 불만족스러워 하고 있다. 만일 쏘프트웨어개발기업체가 충분한 유지정비봉사를 제공해 주지 않는다면 의뢰자는 장래의 제품개발업무를 다른 곳에 맡기게 될것이다. 의뢰자와 쏘프트웨어개발그룹이 같은 기업체에 속해 있다는것으로 하여 장래의 사업견지에서 서로 뒤얽혀 져 있을 때 제품에 불만족해 하는 의뢰자는 모든 정당한 또는 부당한 수단으로써 쏘프트웨어그룹을 믿지 않을수 있다. 이것은 쏘프트웨어개발그룹의 내부와 외부로부터 불신임을 조성하는데로 이어 지며 이로 하여 집단의 해산과 사직을 초래하게 된다.

그러므로 개발되는 매 제품에 있어서 유지정비단계는 쏘프트웨어생산에서 가장 어려운 단계로 되며 흔히 가장 보람이 없는 단계로 된다.

이러한 정황이 어떻게 전환될수 있는가? 경영자들은 유지정비과제를 유지정비를 수행하는데 필요한 모든 기능을 소유하고 있는 프로그람작성자들에게 맡겨야 한다. 경영자들은 가장 높은 수준의 콤퓨터전문가들만이 개발기업체내에서 유지정비에 대한 분담을 받을수 있으며 그들에게 그에 맞는 보수를 지불해야 한다는것을 깨달아야 한다. 만일 관리자측이 유지정비가 하나의 난관으로 되며 훌륭한 유지정비가 개발기업체의 성공을 위한 결정적인 고리로 된다는것은 믿게 되면 유지정비에 대한 태도가 개선되게 될것이다(다음의 《알고 싶은 문제》를 보시오.).

알고 싶은 문제

문헌《*Practical Software Maintenance*》에서 톰 피고스키(Tom Pigoski)는 자기가 플로리다주의 펜사콜라(Pensacola)에 해군성유지정비기업체(Navy maintenance organization)를 어떻게 설립하였는가에 대하여 서술하고 있다. 그의 견해는 만일 취직할 직원들이 자기가 유지정비성원으로 일한적이 있다는것을 사전에 이야기하면 그들은 유지정비에 대하여 긍정적인 태도를 가지고 있다는것이다. 이밖에 그는 모든 직원들이 많은 양성교육을 받게 되며 사업과정에 세계의 모든 곳에 려행할 기회를 가지게 된다는것을 담보함으로써 그들의 사기도 높이려고 시도하였다. 아름다운 해변가는 확실히 그들이 새로 지은 건물에 들었을 때처럼 도움이 되였다.

그러나 유지정비기업체가 사업을 시작한지 6개월도 못되여 모든 직원들이 자기들은 언제 개발과제를 수행하게 되는가를 물었다. 이로 보아 유지정비에 대한 개별적사람들의 태도를 변경시키는것은 극히 어려운 문제인것 같다.

유지정비프로그람작성자들이 현재 직면하여 있는 일부 문제점들은 실례연구에서 고찰하기로 한다.

16. 3. 유지정비의 실례연구

중앙집권적경제를 운영하는 나라들에서는 정부가 농산물의 분배와 판매를 조종한다. 이와 같은 어느 한 나라에서는 복숭아, 사과, 배와 같은 온대지방 파일과 관련한 문제를 온대파실위원회(*Temperate Fruit Committee*; TFC)에 책임지웠다. 어느 날 TFC위원장이 TFC의 활동을 콤퓨터화하기 위하여 어느 한 정부콤퓨터고문을 청하였다. 위원장은 고문에게 정확히 7가지 온대파일 즉 사과, 살구, 벚, 기름복숭아, 복숭아, 배, 추리가 있다는것을 통보하였다. 많지도 적지도 않게 이 7가지 파일에 대해서만 자료기지가 설계되였다. 결국 이것은 세계와 정부고문이 임의의 종류의 확장가능성을 리용하면서 시간과 자금을 랑비하지 않게 하는 방법이였다.

이 제품은 정식으로 TFC에 배포되였다. 1년쯤 지나서 위원장은 그 제품에 책임 있는 유지정비프로그람작성자를 소환하였다.

위원장은 그에게 《키위파일에 대하여 무엇을 알수 있습니까?》라고 물었다. 어리둥절한 프로그람작성자는 《아무것도 모릅니다.》라고 답변하였다. 위원장은 다음과 같이 말하였다. 《좋습니다. 키위파일은 우리 나라에서 금방 재배하기 시작한 파일인것 같은데 TFC가 그에 대하여 책임을 져야 합니다. 제품을 그에 맞게 변경하십시오.》

유지정비프로그람작성자는 고문이 다행히도 위원장의 처음지시를 그대로 수행하지 않았다는것을 발견하였다. 어떤 종류의 장래확장을 허용하게 하는 훌륭한 습관은 너무 뿌리 깊은것이여서 고문은 련관된 자료기지기록들에 여러개의 쓰이지 않는 마당들을 만들어 놓았다. 일정한 항목들을 약간 재배렬함으로써 유지정비프로그람작성자는 8번째 온대파일인 키위를 제품안에 포함시킬수 있었다.

또 한해가 지나갔는데 제품은 잘 동작하였다. 그때에 유지정비프로그람작성자는 또 위원장사무실에 불리워 갔다. 위원장은 기분이 좋은 상태였다. 위원장은 프로그람작성자에게 정부가 농산물의 분배와 판매를 재조직하였다는것을 유쾌한 기분으로 통보하였다. 위원회는 이제는 온대파일뿐이 아닌 그 나라에서 생산되는 모든 파일을 책임지게 되였으며 따라서 제품은 위원장이 프로그람작성자에게 넘겨 준 목록에 있는 26가지의 추가적인 파일들을 포함하도록 변경되여야 하였다. 프로그람작성자는 이러한 변경을 하려면 제품을 처음부터 다시 작성하는것과 거의 맞먹는 시간이 걸린다는것을 말하면서 항의하였다. 위원장은 《허튼 소리 마시오.》라고 말하였다. 그러면서 《당신은 아무런 애로도 없이 키위를 추가하였소. 그러니 그와 같은 일을 26번 반복하시오!》라고 지시하였다.

이 실례로부터 여러가지 중요한 교훈을 찾게 된다.

1. 제품에 확장할수 있는 준비가 되여 있지 않다는 문제점은 유지정비성원이 아니라 개발자로부터 초래되였다. 개발자가 그 제품의 확장능력에 관하여 위원장의 지시를 집행하는데서 오유를 범하였으나 유지정비자가 그 후과를 입었다. 사실 초기의 제품을 개발한 고문이 이 책을 읽지 않는 한 그는 자기의 제품이 성공이

아니라는것을 절대로 깨닫지 못할수 있다. 이것은 유지정비성원이 다른 사람의 오유를 수정할 책임을 지게 된다는 점에서 유지정비에서 제기되는 매우 불쾌한 한가지 측면이다. 문제를 발생시킨 당사자는 다른 임무를 맡았거나 이미 개발기업체를 떠났는데 유지정비프로그람작성자는 여전히 그 시달림을 받고 있는것이다.

2. 의뢰자는 흔히 유지정비가 어려울수 있으며 또는 일부 경우에는 불가능하다는것을 리해하지 못하였다. 의뢰자가 보기에는 새로운 유지정비과제가 이전에 크게 애로가 없이 수행한것과 아무런 차이가 없는것 같은데 유지정비프로그람작성자가 새로운 유지정비과제를 수행할수 없다고 항의하는 경우에 문제가 더욱 심각해 진다.
3. 모든 쏘프트웨어개발은 장래의 유지정비를 반복하여 진행되여야 한다. 만일 고문이 임의의 서로 다른 종류의 파일에 대한 제품을 설계하였더라면 처음에 키위를 추가하고 그다음에 26가지의 다른 종류의 파일을 추가할 때에 아무런 애로도 없었을것이다.

여러번 설명한바와 같이 유지정비단계는 쏘프트웨어생산에서 사활적인 단계이며 최대의 자원을 소비하는 단계이다. 제품개발기간에 개발팀이 제품이 일단 설치된 다음에 그 상품을 책임지게 된다. 유지정비프로그람작성자를 절대로 소홀히 하지 않는것이 본질적인 문제이다.

16. 4. 유지정비의 관리

유지정비단계에서 유지정비의 관리와 관련한 문제들을 이제 고찰하기로 한다.

16. 4. 1. 오유보고서

어떤 제품을 유지정비할 때 첫째로 필요한것은 제품을 변경시키기 위한 기구이다. 교정유지정비 즉 잔류오유의 제거와 관련하여 만일 제품이 부정확하게 동작하는것 같으면 사용자에 의하여 오유보고서(*fault report*)가 작성되여야 한다. 오유보고서는 유지정비프로그람작성자가 보통 어떤 종류의 쏘프트웨어오유로 될수 있는 문제를 바로 잡는데 충분한 정보를 포함하여야 한다.

리상적으로 사용자가 제기하는 모든 오유는 즉시에 수정되여야 한다. 실천적으로 프로그람개발기업체들은 개발 및 유지정비작업에서 나서는 주문잔고들의 처리로 인하여 늘 일손이 모자란다. 만일 어떤 생활비지불명부 관리제품이 지불날자 전날 폭주되거나 직원들에게 생활비를 초과지불하거나 모자라게 지불하는것과 같이 오유가 치명적이라면 즉시적인 정정이 진행되여야 한다. 한편 매 오유보고서는 적어도 즉시적인 예비조사를 받아야 한다.

유지정비프로그람작성자는 우선 오유보고파일을 참고하여야 한다. 오유보고파일은 아직 수정되지 않은 모든 보고서오유들과 그 오유를 에돌아 작업하는것과 관련한 제안들 즉 그 오유가 수정될수 있을 때까지 그 오유에 명백히 책임이 있는 제품의 부분들을 사용자가 우회하게 하는 방법들을 포함하고 있다. 만일 오유가 사전에 보고되였으면 이 오

유보고서파일에 있는 모든 정보들은 사용자에게 주어 져야 한다.

그러나 만일 사용자가 보고한것이 어떤 새로운 오유로 될것 같으면 유지정비프로그람작성자는 그 문제를 연구하고 그 원인과 수정방도를 찾기 위하여 노력하여야 한다. 왜냐하면 누구인가에게 그 쏘프트웨어에 필요한 변경을 진행하도록 임무가 맡겨 질 때까지는 6개월 또는 9개월이 걸리수도 있기때문이다. 프로그람작성자들 특히는 유지정비를 진행하는데 충분히 준비된 프로그람작성자들이 부족하다는 사정으로 미루어 보아 오유가 해결될수 있을 때까지 오유를 그냥 두고 쓰는 방법을 제안하는것이 흔히 그리 위급하지 않는 오유보고서를 처리하기 위한 유일한 방도로 된다.

그다음 유지정비프로그람작성자의 결론이 그 결론에 도달할 때 리용한 목록, 설계, 지도서들과 같은 보충적인 문서작성과 함께 오유보고서에 추가되여야 한다. 유지정비를 담당하고 있는 경영자는 각이한 오유들에 대한 우선권을 부여하면서 이 파일을 규칙적으로 참고하여야 한다. 이 파일에는 또한 완전유지정비와 적응유지정비에 대한 의뢰자의 요구를 포함하여야 한다. 가장 높은 우선권을 가진것이 제품에 대한 다음번 변경으로 될것이다.

어떤 제품에 대한 복제본이 여러 싸이트들에 배포되였을 때 오유보고서의 복제본들은 매 오유가 언제 수정될수 있는가에 대한 평가와 함께 모든 제품사용자들에게 차례차례 배포되여야 한다. 그다음 동일한 오유가 다른 싸이트에서 발생하면 사용자들은 그 오유를 에돌아 작업할수 있는가와 그것이 언제 수정될것인가를 결정하기 위하여 련관된 오유보고서를 참고할수 있다. 매 오유를 즉시에 수정하고 새로운 판본의 제품을 모든 싸이트들에 배포하는것이 물론 더 중요할수 있다. 현재 세계적범위에서 우수한 프로그람작성자들의 부족과 유지정비의 현실성으로 하여 오유보고서의 배포가 아마도 할수 있는 최선의 방도인것 같다.

오유들이 일반적으로 즉시에 수정되지 못하는 또 한가지 리유가 있다. 즉 여러가지 변경을 진행하고 그것들을 모두 시험하고 문서를 변경하고 새로운 판본을 설치하는것이 매 변경을 따로따로 진행하고 그다음 그것을 시험하고 문서화하며 새로운 판본을 설치하는것보다 거의 언제나 값이 더 눅기때문이다. 이것은 만일 매 새로운 판본들이 일정한 수량의 콤퓨터상에 설치되여야 하거나(의뢰기-봉사기망에 많은 의뢰자들이 있는것과 마찬가지로) 그 쏘프트웨어가 다른 싸이트들에서 실행되고 있는 경우에 특히 그러하다. 결국 개발기업체들은 치명적이 아닌 유지정비과제들을 모아 놓았다가 그다음 한개의 그룹으로서 변경을 실현하기를 오히려 더 좋아 한다.

16. 4. 2. 제품에 대한 변경허가

일단 교정유지정비를 수행하기 위한 결심이 서면 어떤 유지정비프로그람작성자에게 실패를 초래한 오유를 결정하고 그것을 수정할 과업이 맡겨 진다. 코드가 변경된후에는 제품전체로써 그 수정내용을 시험하여야 한다(회귀시험). 그다음 그러한 변경을 반영하도록 문서가 갱신되여야 한다. 특히 무엇이 변경되였으며 그것이 누구에 의하여 언제, 무엇때문에 변경되였는가에 대한 자세한 서술을 변경된 매개 모듈의 서두설명문에 추가하여야 한다. 필요한 경우에 설계 또는 명세서도 역시 변경된다. 이와 류사한 단계들이 완전

또는 적응유지정비를 수행할 때에도 진행되는데 유일한 차이점은 완전 및 적응유지정비는 오유보고서에 의해서가 아니라 오유에서의 변경에 의하여 시작된다는것이다.

이 시점에서 필요하다고 보이는것은 새로운 판본을 사용자들에게 배포하는것이다. 그러나 만일 유지정비프로그람작성자가 수정작업을 적당하게 시험하지 않았다면 어떻게 되겠는가? 제품은 배포되기전에 독자적인 그룹에 의해 수행되는 쏘프트웨어품질보증을 받아야 한다. 즉 유지정비SQA그룹성원들은 유지정비프로그람작성자와 같은 경영자에게 보고하지 말아야 한다. SQA가 경영상독자성을 유지하는것이 중요하다(6.1.2)

유지정비가 왜 어려운가 하는 리유는 이미 제시되였다. 이와 같은 리유로 하여 유지정비 역시 실패로 되기 쉽다. 유지정비단계에서의 시험은 어렵고 시간이 많이 랑비되며 따라서 SQA그룹은 시험과 관련하여 쏘프트웨어유지정비의 의의를 파소평가하지 말아야 한다. 일단 새로운 판본이 SQA그룹에 의하여 허가되면 그것은 배포될수 있다.

관리자측이 절차들이 주의 깊게 준수된다는것을 담보하여야 하는 다른 하나의 령역은 기준선기법과 비공식적인 복사가(5.8.2) 리용되는 경우이다. 어떤 프로그람작성자가 클라스 **TaxProvision**을 변경시키려고 한다고 가정하자.

이 프로그람작성자는 그와 관련된 모듈을 고정하고 요구되는 유지정비과제를 수행하는데 필요한 다른 모든 클라스들을 복제한다. 흔히 여기에는 그 제품안의 다른 모든 클라스들이 포함된다. 프로그람작성자는 **TaxProvision**에 필요한 변경을 진행하고 그것들을 시험한다. 그리고 이 변경들을 병합하고 있는 새로운 판본의 **TaxProvision**이 기준선에 설치된다. 그러나 변경된 제품이 사용자에게 배포될 때 그것은 즉시 폭주된다. 잘못한것은 유지정비프로그람작성자가 자기 개인작업공간의 복제본을 리용하여 변경된 판본의 **TaxProvision**을 시험한것이다. 즉 **TaxProvision**의 유지정비를 진행할 때 기준선이였던 다른 클라스들의 복제본들이 출발되였던것이다. 그사이에 어떤 다른 클라스들이 같은 제품에 대하여 작업하는 다른 유지정비프로그람작성자들에 의하여 갱신되였다. 교훈은 명백하다. 즉 어떤 모듈을 설치하기전에 그것은 프로그람작성자의 개인용판본이 아니라 다른 모든 모듈들의 한개의 기준선판본을 리용하여 시험되여야 한다. 이것은 독자적인 SQA그룹을 요구하는 다른 한가지 리유이다. 즉 SQA그룹의 성원들은 프로그람작성자의 개인작업공간에 전혀 접근하지 말아야 한다. 세번째 리유는 어떤 오유에 대한 초기정정 그자체가 그 당시에 70%정도 부정확하다고 평가되였기때문이다[Parnas, 1999].

１６．４．３．유지정비가능성의 담보

유지정비는 어느 한 때만 하는 작업이 아니다. 잘 작성된 제품은 자기 수명이상으로 계렬화된 판본들로 발전하게 된다. 결과 전체 쏘프트웨어개발과정에서 유지정비를 계획하는것이 필요하다. 실례로 설계단계에서 정보은페수법(7.6)이 리용되여야 하며 실현기간에는 장래의 유지정비프로그람작성자들에게 의미가 있는 변수이름들이 선택되여야 한다(14.3). 문서는 완전하고 정확해야 하며 제품의 매 요소모듈들에 대한 현재의 판본을 반영하고 있어야 한다.

유지정비단계에서는 제품의 맨 첫 시기부터 그 제품에 실현해 놓은 유지정비가능성을 손상시키지 않는것이 중요하다. 달리 말하면 쏘프트웨어개발자들이 항상 앞으로 불가

피하게 있게 될 유지정비에 대하여 자각하고 있어야 하듯이 쏘프트웨어유지정비성원들도 항상 장래의 불가피한 유지정비에 대하여 자각하고 있어야 한다. 개발기간에 유지정비가능성에 관하여 제정한 원리들은 유지정비단계에서도 마찬가지로 리용할수 있다.

16.4.4. 반복유지정비문제

제품개발을 좌절시키는 난점들중의 하나는 이동하는 목표문제이다(2.2.1). 개발자가 제품을 구성하는것만큼 빨리 의뢰자는 요구를 변경시킬수 있다. 이것은 개발팀을 좌절시킬뿐아니라 빈번한 변경으로 인하여 제품이 불충분하게 구성될수 있다. 이밖에 이와 같은 변경들에 드는 자금이 제품의 비용에 추가되게 된다. 리론적으로 이에 대처하기 위한 방도는 신속원형을 작성하는것부터 시작하는것이다. 그러면 의뢰자가 요구를 아무리 자주 변경시켜도 문제로 되지 않는다. 일단 의뢰자가 최종적으로 만족하면 명세서가 승인되고 제품 그자체가 구성된다. 실천적으로 그 무엇도 요구들이 최종적으로 승인된 다음날로 의뢰자들이 그것을 또 변경시키는것을 가로막을수 없다. 이러한 정황에서 원형작성의 기본우점은 의뢰자들에게 작업모형을 줌으로써 변경의 회수와 빈도수를 줄일수 있다는것이다. 그러나 만일 의뢰자가 기꺼히 돈을 지불하기를 원한다면 그 무엇도 매주 월요일과 화요일마다 요구들이 변경되는것을 막을수는 없다.

문제는 유지정비단계에서 악화된다. 어떤 완성된 제품이 많이 변경되면 될수록 그 제품은 초기의 설계로부터 더욱더 리탈하게 될것이며 장래의 변경을 더욱더 어렵게 만들것이다. 반복유지정비정황하에서 문서는 보통때보다 훨씬 더 믿을수 없게 될수 있으며 회귀시험파일들은 갱신되지 못할수도 있다. 만일 유지정비가 훨씬 더 많이 진행되면 제품전체를 현재의 판본을 하나의 원형으로 리용하여 처음부터 완전히 작성하여야 할수도 있다. 이동목표문제는 명백히 관리자측과 관련한 문제이다. 리론적으로 만일 관리자측이 의뢰자와 충분히 결합되여 있으며 프로젝트를 실행하는 초기에 문제를 설명한다면 요구들은 원형이 인입된 때로부터 제품이 배포될 때까지 고정될수 있다. 또한 완전유지정비에 대한 매 요청이 진행된후에 요구들은 3개월 또는 1년동안 고정될수 있다. 실천적으로 이런 방식으로는 일하지 못한다.

실례로 만일 의뢰자가 우연히 회사의 사장으로 되고 개발기업체는 그 회사의 정보체계부서라면 사장은 실지로 매주 월요일과 화요일에 변경을 주문할수 있으며 그런 변경들은 실현될것이다. 《비용을 부담하는 사람에게 결정권이 있다.》는 옛날 속담은 공교롭게도 이 정황에 너무도 적합하다. 아마 정보체계의 부책임자가 할수 있는 최선의 방법은 사장에게 반복유지정비가 제품에 주는 영향은 이후의 유지정비가 제품의 통합에 위험을 가져 올 때마다 그 완성된 제품을 단순히 다시 작성하게 하는것이라는것을 설명하려고 시도하는것일수 있다.

요구되는 변경이 천천히 실현된다는것을 담보함으로써 추가적인 유지정비를 방해하려고 하는것은 이와 관련 있는 직원을 유지정비를 보다 빨리 할수 있는 다른 직원으로 교체하는 결과만을 초래할것이다. 간단히 말하여 반복변경을 요구하는 사람이 충분한 권한을 가진 사람이라면 이동목표와 관련한 이 문제를 해결할 방도는 더는 없다.

１６．５．객체지향쏘프트웨어의 유지정비

객체지향파라다임을 리용할것을 권고한 한가지 리유는 그것이 유지정비를 촉진시킨다는것이다. 결국 하나의 객체는 어떤 프로그람의 하나의 독립적인 단위이다. 보다 명확하게 말하면 잘 설계된 객체는 교갑화(*encapsulation*)로 알려 진 개념적독립성을 나타낸다(7.4). 실세계의 부분과 관계되는 제품의 매개 측면은 객체 그자체로 국부화된다. 이밖에 객체는 물리적독립성을 나타낸다. 즉 실현세부들은 그 객체의 밖에서 볼수 없다는것을 담보하기 위하여 정보은페수법을 인입한다(7.6). 허용되는 정보전달의 유일한 형식은 어떤 특정한 객체를 호출하기 위하여 그 객체에 통보를 보내는것이다.

결과 론리적으로 보면 다른 두가지 리유로 하여 객체를 유지정비하는것이 쉬워 질것이다. 첫째로, 개념적독립성은 어떤 특정한 유지정비목표를 달성하기 위하여 제품의 어느 부분이 변경되여야 하며 그것이 확장유지정비인가 아니면 교정유지정비인가를 쉽게 결정하게 할것이라는것을 의미한다. 둘째로, 정보은페는 어떤 객체에 가해 질 변경이 그 객체의 밖에서 충돌하지 않을것이며 그로 인하여 회귀오유의 수가 크게 감소된다는것을 담보한다.

그러나 실천적으로 정황은 이처럼 랑만적인것은 아니다. 사실 객체지향쏘프트웨어에 특정한 세가지 장애가 존재한다. 그중 한가지 문제는 적당한 CASE도구를 사용하는것을 통하여 해결될수 있지만 다른것들은 처리하기가 쉽지 않다. 먼저 그림 16−2에 보여 준 C++클라스계층도를 고찰하자. 방법 displayNode는 클라스 **UndirectedTree**에서 정의되고 클라스 **DirectedTree**에 의하여 계승되며 그다음 클라스 **RootedTree**에서 재정의된다. 재정의된 판본은 클라스 **BinaryTree**와 클라스 **BalancedBinaryTree**에 의해 계승되며 클라스 **BalancedBinaryTree**에서 리용된다. 그러므로 유지정비프로그람작성자는 클라스 **BalancedBinaryTree**를 리해하기 위하여 완전한 계승계층도를 연구하여야 한다.

보다 불리한 경우에 계층도가 그림 16−2의 선형형식으로 현시되지 않을수도 있지만 일반적으로 전체 제품에 전개되여 있다. 그러므로 클라스 **BalancedBinaryTree**에서 displayNode가 무슨 역할을 하는가를 리해하기 위하여서는 유지정비프로그람작성자가 제품의 중요부분들을 통독하여야 할수 있다. 이것은 이 절의 서두에서 서술한 《독립적인》객체와는 철저한 차이가 있다. 이 문제의 해결은 간단하다. 즉 적당한 CASE도구를 리용하면 된다. C++콤파일러가 클라스 **BalancedBinaryTree**의 실례내에서 displayNode의 판본을 정확히 결정할수 있기때문에 프로그람작성작업대는 어떤 클라스의 《단조로운》판본 즉 임의의 이름재작성 또는 재정의를 병합함으로써 직접적으로 또는 간접적으로 계승되는 모든 명백한 특성들을 가지고 있는 클라스의 정의를 제공하여 준다. 그림 16−2 에서 클라스 **BalancedBinaryTree**의 단조로운 형식은 클라스 **RootedTree**로부터 displayNode의 정의를 포함한다. 단조로움을 실현하기 위한 CASE도구들은 C++ 또는 Java에만 국한되여 있지 않다. 실례로 에필(Eifell)은 이러한 목적을 위하여 명령 **flat**를 포함하고 있다[Meyer, 1990].

객체지향언어를 리용하여 실현된 제품의 유지정비에서의 두번째 장애는 해결하기가 그리 쉽지 않다. 이 문제는 7.8에서 설명한 개념들인 다형성과 동적맺기의 결과로서 발생한다. 이 절에서 한가지 실례가 주어 졌는데 여기에서는 기초클라스를 세개의 부분클라스 즉 **DiskFileClass**, **TapeFileClass**, **DisketteFileClass**들과 함께 **FileClass**로 이름 지었다.

```
{
   …
   void dislayNode ( Node a);
   …
} // 클라스 UndirectedTree

class DirectedTree: public UndirectedTree
{
   …
} // 클라스 DirectedTree

class RootedTree: public DirectedTree
{
   …
   void displayNode ( Node a);
   …
} // 클라스 RootedTree

class BinaryTree: public RootedTree
{
   …
} // 클라스 BinaryTree

class BalancedBinaryTree: public BinaryTree
{
   Node        hhh;
   DisplayNode (hhh);
} // 클라스 BalancedBinaryTree
```

그림 16-2. 클라스계층의 C++실현

이것은 그림 7-33에 제시되였는데 편리를 도모하기 위하여 이 장에서 그림 16-3으로 다시 제시하였다. 기초클라스 **FileClass**에서 하나의 무의미한(**abstract** 또는 **virtual**) 방법 open이 선언된다. 그다음 이 방법의 특정한 실현이 세개의 부분클라스들에 각각 반영된다. 즉 매 방법은 그림 16-3에 보여 준바와 같이 동일한 이름 open으로 주어 진다. myFile이 하나의 객체 즉 FileClass의 하나의 실례로 선언되고 유지정비되여야 할 코드안에 통보 myFile.open( )이 포함된다고 가정하자. 다형성과 동적맺기의 결과로서 실행시에 myFile은 **FileClass**에서 파생된 세개의 클라스 즉 디스크파일, 테프파일, 자기원판파일의 한 성원으로 될수 있다. 일단 실시간체계가 그 클라스가 어디서 파생되였는가를 결정하면 open의 적당한 판본이 호출된다.

이것은 유지정비에 불리한 결과를 가져 올수 있다. 만일 유지정비프로그람작성자가 코드에서 호출 myFile.open( )을 만나게 되면 제품에서 이 부분을 리해하기 위하여 myFile이 세개 부분클라스의 매개의 한 실례로 되는 경우 무슨 현상이 발생하겠는가를 고찰하여야 한다. CASE도구는 여기서 도움이 될수 없다. 왜냐하면 일반적으로 정적방법들을 리

용하여 동적맺기를 결정할 방도가 없기때문이다.

여러개의 동적맺기들가운데서 어느것이 특수한 정황모임안에서 실지로 발생하였는가를 결정하기 위한 유일한 방도는 해당한 코드를 어떤 콤퓨터상에서실행시키든가 수동적으로 추적하면서 코드를 추적하는것이다. 다형성과 동적맺기는 사실상 객체지향제품의 개발을 촉진시키는 매우 강력한 객체지향기술측면들이다. 그러나 그것들은 유지정비프로그람작성자가 실행시에 발생할수 있는 다양한 형태와 가능한 결합들을 조사하고 그로부터 여러개의 각이한 방법들중에서 어느것이 코드안의 그 개소에서 호출될수 있는가를 결정하도록 함으로써 유지정비에 유해로운 영향을 줄수 있다.

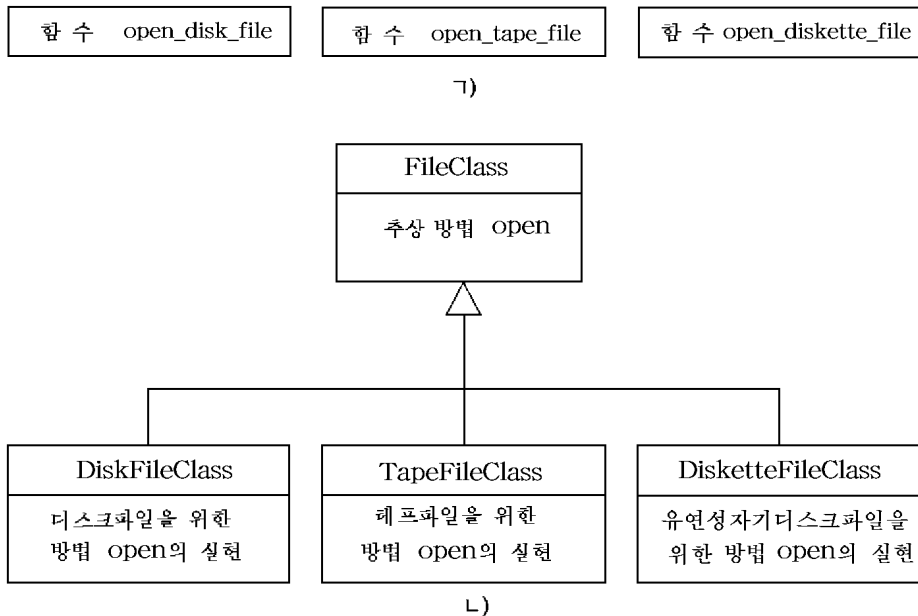



그림 16-3. 도출클라스 **DiskFileClass**, **TapeFileClass**, **DisketteFileClass**를 가진 기초클라스 **FileClass**의 정의

세번째 문제는 계승의 결과로써 발생한다. 어떤 특정한 기초클라스가 새 제품의 설계에서 요구되는 전부는 아닌 대부분의 과제를 수행한다고 가정하자.

이제 하나의 파생클라스를 정의하자. 즉 이 클라스는 많은 점에서 기초클라스와 같지만 새로운 특성들이 추가될수 있으며 현존 특성들이 다시 이름 지어 지거나 다시 실현되여 압축되거나 기타 다른 방식으로 변경될수 있다. 더우기 이러한 변경들은 기초클라스 또는 임의의 기타 파생클라스들에 아무런 영향을 주지 않으면서 진행될수 있다. 그러나 이제 기초클라스자체가 변경된다고 가정하자. 만일 그렇게 되면 모든 파생클라스들이 같은 방식으로 변경된다. 달리 말하면 계승의 우점은 새로운 잎들이 계승나무에서 그 어떤 다른 클라스들을 변경시키지 않으면서 계승나무(만일 실현언어가 C++에서처럼 다중계승을 지원한다면 그라프로 된다.)에 추가될수 있다는것이다. 결국 계승은 개발에는 긍정적인 영향을 주지만 유지정비에는 부정적영향을 줄수 있는 객체지향수법의 또 한가지 특성이다.

1 6. 6. 유지정비기능 대 개발기능

이 장의 앞에서 유지정비에 요구되는 기능들에 대하여 많이 설명하였다. 교정유지정비에 있어서는 대규모제품의 실패원인을 결정하는 능력이 필수적으로 요구된다. 그러나 제품유지정비에서는 이와 반대로 이러한 능력이 요구되지 않는다. 이 기능은 통합과 제품시험 전 기간에 리용된다. 또 한가지 사활적인 기능은 적당한 문서작성을 진행하지 않고 효과적으로 기능을 수행할수 있는 능력이다. 게다가 문서작성은 통합과 제품시험이 진행되는 기간에는 좀처럼 완성되지 않는다. 또한 강조해야 할것은 명세서작성, 설계, 실현과 통합과 관련한 기능들이며 시험이 적응 및 완전유지정비에서 본질적으로 중요하다는것이다. 이러한 활동들은 역시 개발공정에서도 수행되며 그 매개는 그것이 정확하게 수행되여야 한다면 전문화된 기능들을 요구한다.

달리 말하면 유지정비프로그람작성자에게 필요한 기능들은 쏘프트웨어생산의 다른 단계들에 전문적으로 봉사하는 쏘프트웨어전문가들에게 필요한 기능들과 방법상 아무러한 차이도 없다. 중요한 점은 유지정비프로그람전문가들이 다양한 령역들에 단순히 숙련될것이 아니라 그 모든 령역에 고도로 숙련되여야 한다는것이다. 비록 평범한 쏘프트웨어개발자가 설계나 시험과 같은 쏘프트웨어개발의 한 령역에 전문화될수 있다고 하여도 쏘프트웨어유지정비성원은 사실상 쏘프트웨어생산의 모든 령역에 대한 전문가로 되여야 한다. 결국 유지정비는 개발이나 다름이 없거나 또는 그이상으로 될수 있다.

1 6. 7. 역 공 학

때때로 지적한바와 같이 유지정비에 리용할수 있는 유일한 문서는 원천코드 그자체이다(이러한 경우는 기존체계(*legacy system*) 즉 현재 리용하고 있지만 15년 또는 20년전에 개발된 쏘프트웨어를 유지정비할 때에 매우 자주 발생한다.). 이러한 정황하에서 코드를 유지정비하는것은 매우 어려울수 있다. 이러한 문제를 처리하기 위한 한가지 방도는 원천코드로부터 출발하여 설계문서나 지어는 명세서를 다시 작성하려고 시도하는것이다. 이러한 과정을 역공학*(reverse engineering*)이라고 한다.

CASE도구들이 이 과정을 도울수 있다. 한가지 가장 간단한 실례는 기묘한 인쇄프로그람(5.6)인데 이것은 코드를 보다 명백히 현시할수 있도록 도와 줄수 있다. 다른 도구들은 원천코드로부터 직접 흐름도나 UML선도과 같은 선도들을 구성하는데 이러한 시각적인 수단들은 설계회복과정을 도울수 있다. 일단 유지정비팀이 설계를 재구성하였다면 두가지 가능성이 존재하게 된다. 하나의 대안은 명세서를 재구성하고 재구성된 명세서를 변경하여 필요한 변경들을 반영하며 제품을 보통 방식으로써 재실현하는것이다(역공학의 범위내에서 명세서의 작성으로부터 설계를 통하여 코드작성에로 나아가는 일반적인 개발공정을 정공학(*forward engineering*)이라고 부른다. 역공학에 뒤이은 정공학을 때로 재공학(*reengineering*)이라고 부른다.). 실천적으로 명세서의 재구성은 매우 어려운 과제이다. 흔히 재작성된 설계는 변경되며 변경된 설계는 그다음 정공학적으로 실현된다. 유지정비단계에서 흔히 실현되는 련관된 활동은 재구성(*restructuring*)이다. 역공학은 제품을 보다 낮은 준위의 추상화로부터 보다 높은 준위의 추상화에로 례하면 코드로부터 설계에로 끌어

올린다. 정공학은 보다 높은 준위의 추상화로부터 보다 낮은 준위의 추상화에로 제품을 끌어 내린다. 그러나 재공학은 같은 준위에서 진행된다. 재구성은 제품의 기능을 변경시키지 않고 제품을 개선하는 과정이다. 기묘한 인쇄프로그람은 재구성의 한가지 형태이다. 그러므로 코드는 비구조화된 형식으로부터 구조화된 형식으로 전환된다. 일반적으로 재구성은 원천코드(또는 설계 지어는 자료기지)를 보다 쉽게 유지정비하기 위하여 진행된다.

만일 원천코드가 류실되고 리용할수 있는것이 제품의 실행판본인 경우에 정황은 더욱 불리해 진다. 얼핏 보기에는 원천코드를 재생하기 위하여 유일하게 가능한 방도는 역아쎔블러를 리용하여 역아쎔블리어코드를 생성하며 원래의 고급언어코드들을 재생할수 있게 하는 코드(이것을 역콤파일러라고 부를수 있다.)를 구성하는것이다. 이 방법에는 사실상 극복할수 없는 여러가지 문제점들이 동반되게 된다.

1. 변수의 이름이 본래의 콤파일의 영향으로 류실될수 있다.
2. 대부분의 콤파일러들은 어떤 방법으로 코드를 최량화하는데 이것은 원천코드를 재생하려고 시도하는것을 매우 어렵게 한다.
3. 아쎔블리어에서 순환과 같은 구조는 원천코드에서 여러가지 서로 다른 가능한 구조들에 대응할수 있다.

그러므로 실천적으로 현존 제품은 검은통으로서 취급되며 역공학은 현재제품의 거동으로부터 명세서를 도출해 내는데 리용된다. 재구성된 명세서들은 필요할 때 변경되며 제품의 새로운 판본은 이 명세서로부터 정공학적수법으로 실현된다.

16. 8. 유지정비단계에서의 시험

제품이 개발되는 동안에 개발팀의 많은 성원들이 제품전반에 대한 폭넓은 견해를 가지게 되지만 콤퓨터산업에서의 인재들의 빠른 교체로 인하여 유지정비에 참가하는 유지정비팀성원들은 초기개발에 참가하지 못하였을수도 있다. 그러므로 유지정비성원들은 제품을 제각기 련관된 모듈들의 모임으로 보는 경향이 있으며 한개 모듈에 대한 변경이 한개 또는 그이상의 다른 모듈들에 심각한 영향을 주며 나아가서 제품전반에 영향을 주게 된다는것을 일반적으로 깨닫지 못한다. 비록 유지정비성원이 제품의 모든 측면을 리해하려고 애 쓴다고 하여도 제품을 수정하거나 확장하는데서 난관은 이 목적을 달성하기 위하여 요구되는 상세한 연구에 시간을 돌릴수 없다는것이다. 더우기 많은 경우에 이러한 리해를 진행할 때 도움이 되도록 리용할수 있는 문서가 적거나 전혀 없다. 이와 같은 난관을 최소로 줄이기 위한 한가지 방법은 회귀시험의 리용 즉 변경된 제품이 여전히 정확히 동작한다는것을 담보하기 위하여 선행한 시험실례들과 대비하여 변경된 제품을 시험하는것이다. 이러한 리유로 하여 모든 시험실례들을 예상되는 결과와 함께 기계가독한 형식으로 저장하는것이 사활적인 문제로 된다. 제품에 대한 변경의 결과로 일부 저장된 시험실례들이 변경되여야 할수 있다.

실례로 만일 세금법제정의 결과로 로임원천과세률이 변화되면 로임지불명단관리제품의 정확한 출력은 원천과세를 포함하고 있는 매 시험실례에 관하여 변화될것이다. 이와 류사하게 만일 위성관측결과 어떤 섬의 위도와 경도가 수정된다면 그 섬의 좌표를 리용

하여 비행기의 위치를 계산하는 어떤 제품의 정확한 출력도 따라서 변화될것이다. 수행된 유지정비에 따라서 일부 정당한 시험실례들이 부정당하게 되여 버릴것이다. 그러나 저장된 시험실례들을 교정하기 위하여 진행할 계산들은 유지정비가 정확하게 수행되였다는것을 시험하기 위한 새로운 시험자료들을 구성하기 위하여 진행하여야 할 계산과 본질에 있어서 같다. 그러므로 시험실례들과 예상되는 결과들을 저장하고 있는 파일을 유지정비할 때 그 어떤 추가적인 작업도 필요 없다. 회귀시험은 시간을 랑비한다고 론의될수 있다. 왜냐하면 회귀시험은 완성제품의 제품유지정비과정에 변경된 모듈들과 명백히 아무런 관련도 없는 많은 시험실례들에 대하여 재시험될것을 요구하기때문이다. 여기서 《명백히》라는 단어가 결정적의미를 가진다. 미처 의식하지 못한 유지정비의 측면영향의 위험(즉 회귀시험의 인입이 실패한다.)은 매우 커서 이러한 론증을 무의미하게 만든다. 즉 회귀시험은 그 어떤 정황에서도 유지정비의 본질적인 한가지 측면으로 된다.

16. 9. 유지정비단계에서의 CASE도구

유지정비프로그람작성자가 어떤 모듈이 갱신될 때마다 각이한 재판본번호들을 수동적으로 추적하여 다음번 판본번호들을 할당하리라고 생각하는것은 부당한 일이다. 조작체계가 판본조종기능을 포함하고 있지 않는 한에서는 UNIX도구 ***sccs***(source code control system)[Rochkind, 1985], ***rcs***(revision control system)[Tichy, 1985] 또는 ***c v s***(concurrent versions system)[Loukides and Oram, 1997]과 같은 어떤 판본조종도구가 요구된다. 제5장에서 서술한 동결기법의 수동조종이든가 또는 재판본이 적당하게 갱신된다는것을 담보하는 기타 수동적인 방법들을 기대하는것도 역시 부당한 일이다. 필요한것은 하나의 구성조종도구이다. 시장에서 판매되는 한가지 전형적인 도구는 CCC(*change and configuration control*)이다. 비록 쏘프트웨어개발기업체가 완전한 구성조종도구를 구입하려고 하지는 않는다고 하여도 최소한 구축도구(*build tool*)는 어떤 판본조종도구와 결합하여 리용되여야 한다. 유지정비단계에서 사실상 본질적역할을 하는 또 다른 부류의 CASE도구들은 아직 수정되지 않는 발견오유들에 대한 기록을 유지하게 하는 오유추적도구이다.

16.7에서 역공학과 재공학을 지원할수 있는 몇가지 부류의 CASE도구들을 서술하였다. 제품구조에 대한 시각적현실을 생성하는것으로써 지원을 주는 이와 같은 도구들로서는 Battlemap, Teamwork, Bachman Reengineering Product Set들을 실례로 들수 있다.

유지정비는 어렵고 실패하기 쉽다. 관리자측이 최소한 할수 있는것은 유지정비팀에 효과적이고 또 효과적인 제품유지정비에 필요한 도구들을 제공하여 주는것이다.

16. 10. 유지정비단계에서의 척도

유지정비단계에서 진행되는 활동은 본질에 있어서 명세작성, 설계, 실현, 통합, 시험, 문서작성이다. 그러므로 이러한 활동들을 재기 위한 척도들은 유지정비단계에서도 그대로 리용할수 있다. 실례로 14.8.2의 복잡성척도는 높은 복잡도를 가진 어떤 모듈이 회귀오유를 포함하는 후보로 될것 같다고 하는 점에서 유지정비와 관련된다. 이와 같은 모듈

을 변경할 때에는 특별히 주의를 돌려야 한다.

이밖에 유지정비단계에 특정한 척도들로서는 보고된 오유의 총 개수와 이 오유들의 엄중성과 종류에 따르는 분류와 같은 쏘프트웨어오유보고서와 관련된 측도들을 들수 있다. 이밖에 오유보고서의 현재상태와 관련한 정보들도 요구된다. 실례로 2002년에 13개의 치명적오유들이 보고되고 수정되였다는것과 그 해에 2개의 치명적오유만이 보고되고 수정되지 않았다는것사이에는 상당한 차이가 있는것이다.

16.11. 항공음식전문회사 실례연구 : 유지정비단계

여러가지 오유들이 이 실례연구의 원천코드에서 발견되였다. 그중 3개를 발견하여 정정하는것을 련습문제로서 남겨 둔다(문제 16.11~16.13까지).

16.12. 유지정비단계에서의 난관

유지정비단계에서의 기본 난관들은 1.3의 《알고 싶은 문제》에서 자세히 설명하였다. 간단히 말하여 고전적쏘프트웨어공학은 선개발후 유지정비(*development-then-maintenance*)로써 설명할수 있다. 즉 먼저 제품의 령상태로부터 개발되여 의뢰자에게 배포된다. 그때로부터 제품에 대하여 유지정비가 진행된다. 이와 같은 선개발후유지정비모형은 다음과 같은 IEEE규격610.12의 정의와 일치한다. 즉 《쏘프트웨어가 의뢰자에게 배포되기전에 수행되는 모든 쏘프트웨어생산활동은 개발로 간주되며 배포이후의 모든 활동들은 유지정비를 구성한다.》[IEEE 610.12, 1990]

선개발후유지정비모형은 오늘날 비현실적이다.

1. 의뢰자의 요구는 제품이 배포되기전에 자주 변한다.
2. 오유들은 흔히 제품이 배포되기전에 수정되여야 한다.
3. 령상태로부터의 개발은 점차 드물어 지고 있다. 반대로 제품의 대부분은 재리용된 요소들밖에서 구성되며 이러한 재리용된 요소들은 그것들이 새 제품에서 리용될수 있기전에 자주 변경될 필요가 있다.

이와 같은 세가지 리유로 하여 제품은 거의 언제나 배포되기전에 변경되여야 하며 이러한 변경들을 진행할 때에 수행된 활동들은 고전적유지정비에서의 활동과 구별할수 없다.

유지정비의 실제의미는 ISO/IEC규격12207에서 제정되였는데 여기에서는 유지정비공정은 《쏘프트웨어가 어떤 문제나 개선 또는 적응에서의 필요로 인하여 코드와 련관된 문서에 의하여 변경을 거쳐야 할》 때 작용된다고 서술하고 있다[ISO/IEC 12207, 1995].

쏘프트웨어공학집단의 보다 광범한 계층이 《유지정비》는 쏘프트웨어생명주기의 전반에 걸쳐 수행되는 활동이라는것을 인정하게 될것이다. 이것은 유지정비단계뿐아니라 전반적 쏘프트웨어개발공정에 중요한 개선을 가져 오게 될것이다.

요 약

유지정비는 중요하고도 어려운 쏘프트웨어활동이다(16.1과 16.2). 이것을 16.3의 실례연구로써 례증한다. 반복유지정비문제를 비롯하여 유지정비의 관리와 관련된 문제들을 서술한다(16.4). 객체지향쏘프트웨어의 유지정비는 16.5에서 론의된다. 유지정비프로그람작성자에게 요구되는 기능은 개발자에게 필요한 기능과 같다. 차이는 개발자는 쏘프트웨어개발공정의 한 측면에 전문화될수 있으며 반면에 유지정비성원은 쏘프트웨어생산의 모든 측면에서의 전문가로 되여야 한다는것이다(16.6). 역공학에 대한 설명은 16.7에 서술되였다. 유지정비단계에서의 시험(16.8)과 유지정비단계에서의 CASE도구(16.9)들에 대한 설명을 진행한다. 유지정비단계의 척도는 16.10에서 론의된다. 항공음식전문회사의 실례연구에 대한 유지정비는 16.11에서 제시되였는데 련습으로 남겨 두었다. 이 장은 유지정비단계에서의 난관에 대한 론의로 결속된다(16.12).

보 충

유지정비에 대한 고전적인 정보원천은 문헌 [Lientz and Swanson, 1980]이다. 유지정비에 대한 유용한 자료는 문헌 [Longstreet, 1990]에서 볼수 있다. 문헌 [Basili, 1990]은 재리용과정과 마찬가지로 유지정비에 대한 흥미 있는 견해를 서술하고 있다. 회귀시험은 문헌 [Rothermel and Harrold, 1996]에서 분석하였다. 그 비용상효과성에 대한 예견에 대하여서는 문헌 [Rosenblum and Weyuker, 1997]에서 서술하고 있다. 공업환경에서의 회귀시험은 문헌 [Onoma, Tsai, Poonawala, and Suganuma, 1998]에서 론의하였다. 재공학에 대한 계획작성은 *IEEE Software* 1995년 1월호에 있는 재공학에 관한 많은 기사들중의 하나인 문헌 [Sneed, 1995]에서 서술하고 있다. 재공학의 비용과 리익성에 대하여서는 문헌 [Adolph, 1996]에서 론의하였다. 문헌 [Charette, Adams, and White, 1997]은 유지정비의 기본틀거리안에서 위험성관리를 서술하고 있다. 그리고 문헌 [von Mayrhauser and Vana, 1997]은 큰 규모제품의 유지정비를 위한 여러가지 프로그람리해기구를 서술하고 있다. 아주 성공적인 재공학프로젝트의 륜곽에 대하여 문헌 [Teng, Jeong, and Grover, 1998]에서 서술하여 주고 있다. 정보체계의 유지정비는 문헌 [Bisbal, Lawless, Wu, and Grimson, 1999]에서 서술하여 주고 있다. 유지가능성의 범위안에서 척도의 리용에 대하여서는 문헌 [Banker, Datar, Kemerer, and Zweig, 1993, and Henry, Henry, Kafura, and Matheson, 1994]에서 론의하였다. 객체의 리용에서 유지정비의 영향은 문헌 [Henry and Humphrey, 1990, and Mancl and Havanas, 1990]에서 서술하고 있다. 특정한 객체지향제품의 유지정비는 문헌 [Lejter, Meyers, and Reiss, 1992, and Wilde, Matthews, and Huitt, 1993]에서 서술하여 주고 있다. 쏘프트웨어유지정비와 관련한 론문은 *Commnunication of the ACM* 1994년 5월호에서 찾아 볼수 있다. *IEEE Software* 1998년 7월/8월호에는 기존체계에 대한 많은 기사들이 들어 있으며 여기에는 특히 문헌 [Rugaber and White, 1998]이 있다. 쏘프트웨어유지정비에 관한 대회회보는 유지정비의 모든 측면에 대하여 기초로 되는 폭넓은 정보원천으로 되고 있다.

문 제

16.1. 당신은 무엇때문에 오유가 쏘프트웨어유지정비가 쏘프트웨어개발보다 못하다고 간주하는데로부터 생긴다고 생각하는가?

16.2. 콤퓨터에 비루스가 있는가를 결정하는 제품을 고찰하자. 이러한 제품에서 왜 그 모듈의 대부분이 다중변화될것 같은가를 설명하시오. 그 결과 발생하는 문제는 어떻게 해결될수 있는가?

16.3. 문제 8.7의 자동서고순환체계에 대하여 문제 16.2를 반복하시오.

16.4. 은행계산서가 정확한가를 시험하는 문제 8.8의 제품에 대하여 문제 16.2를 반복하시오

16.5. 문제 8.9의 자동출납기에 대하여 문제 16.2를 반복하시오.

16.6. 당신은 대규모쏘프트웨어개발기업체안의 유지정비를 책임진 경영자이다. 당신은 새로운 직원을 채용할 때 무슨 자질을 중요시하겠는가?

16.7. 직원이 한명인 쏘프트웨어개발기업체에서 유지정비의 의미는 무엇인가?

16.8. 당신은 어떤 콤퓨터화된 오유보고서파일을 구성할데 대한 요청을 받았다. 그 파일에 무슨 종류의 자료를 저장하겠는가? 당신의 도구는 무슨 종류의 질문들에 답변할수 있는가? 또 무슨 종류의 질문들에 답변할수 없는가?

16.9. 당신은 Ye Olde Fashioned Software Corperation(문제 15.6)의 쏘프트웨어유지정비부책임자로부터 한통의 편지를 받았다. 편지에서는 가까운 앞날에 Olde Fashioned가 수천만행에 달하는 COBOL코드를 유지정비하게 된다는것을 지적하고 이러한 유지정비를 위한 CASE도구와 관련하여 당신의 권고를 부탁하고 있다. 당신은 어떻게 답변하겠는가?

16.10. (과정안상 목표) 브로드랜즈지역 아동병원(부록 1)의 가족면회분파에서 제공하는 무임수송 및 숙박봉사는 현재 부모들이 브로드랜즈의 926km 반경내에 살고 있는 어린이들에게 제한되여 있다. 이제 이 제한이 제거되게 된다. 문제 15.10의 제품에 대하여 무슨 변경을 하여야 하는가? 당신의 대답을 문제 1.11에 준 대답과 비교하시오.

16.11. (실례연구) 일단 어떤 좌석이 한 승객에게 할당되였으면 다른 승객에게는 배당될수 없도록 15.13의 실현을 정정하시오.

16.12. (실례연구) 항공음식전문회사 실례연구에 13종류의 특별식사가 있다. 그러나 15.13의 실현에는 프로그람작성자가 12종류만 있다고 생각하게 할것 같은 개소들이 있다. 그 실현을 적당히 수정하시오.

16.13. (실례연구) 15.13의 실현은 사용자가 식사품질을 입력할 때 그 값이 1부터 5사이에 있다는것을 검열하지 않는다. 이 오유를 수정하시오.

16.14. (실례연구) 15.13의 실현에서 매 요소들의 중심맞추기를 조절함으로써 모든 보고서들의 미학적인 표현방식을 개선하시오.

16.15. (실례연구) 15.13의 실현에서 차림표구동입력부분프로그람을 도형사용자대면부로 교체하시오.

16.16. (쏘프트웨어공학독본) 교원은 문헌 [Rugaber and White, 1998]의 복제본을 배포할것이다. 당신은 기존체계를 유지정비하는데서 가장 큰 난관이 무엇이라고 보는가?

참 고 문 헌

[Adolph, 1996] W. S. ADOLPH, "Cash Cow in the Tar Pit: Reengineering a Legacy System," *IEEE Software* **13** (May 1996), pp. 41–47.

[Banker, Datar, Kemerer, and Zwieg, 1993] R. D. BANKER, S. M. DATAR, C. F. KEMERER, AND D. ZWIEG, "Software Complexity and Maintenance Costs," *Communications of the ACM* **36** (November 1993), pp. 81–94.

[Basili, 1990] V. R. BASILI, "Viewing Maintenance as Reuse-Oriented Software Development," *IEEE Software* **7** (January 1990), pp. 19–25.

[Bisbal, Lawless, Wu, and Grimson, 1999] J. BISBAL, D. LAWLESS, B. WU, J. GRIMSON, "Legacy Information Systems: Issues and Directions," *IEEE Software* **16** (September/October 1999), pp. 103–11.

[Charette, Adams, and White, 1997] R. N. CHARETTE, K. M. ADAMS, AND M. B. WHITE, "Managing Risk in Software Maintenance," *IEEE Software* **14** (May/June 1997), pp. 43–50.

[Henry and Humphrey, 1990] S. M. HENRY AND M. HUMPHREY, "A Controlled Experiment to Evaluate Maintainability of Object-Oriented Software," *Proceedings of the IEEE Conference on Software Maintenance,* San Diego, CA, November 1990, pp. 258–65.

[Henry, Henry, Kafura, and Matheson, 1994] J. HENRY, S. HENRY, D. KAFURA, AND L. MATHESON, "Improving Software Maintenance at Martin Marietta," *IEEE Software* **11** (July 1994), pp. 67–75.

[IEEE 610.12, 1990] "A Glossary of Software Engineering Terminology," IEEE 610.12-1990, Institute of Electrical and Electronic Engineers, New York, 1990.

[ISO/IEC 12207, 1995] "ISO/IEC 12207:1995, Information Technology—Software Life-Cycle Processes," International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995.

[Lientz and Swanson, 1980] B. P. LIENTZ AND E. B. SWANSON, *Software Maintenance Management: A Study of the Maintenance of Computer Applications Software in 487 Data Processing Organizations,* Addison-Wesley, Reading, MA, 1980.

[Lientz, Swanson, and Tompkins, 1978] B. P. LIENTZ, E. B. SWANSON, AND G. E. TOMPKINS, "Characteristics of Application Software Maintenance," *Communications of the ACM* **21** (June 1978), pp. 466–71.

[Lejter, Meyers, and Reiss, 1992] M. LEJTER, S. MEYERS, AND S. P. REISS, "Support for Maintaining Object-Oriented Programs," *IEEE Transactions on Software Engineering* **18** (December 1992), pp. 1045–52.

[Longstreet, 1990] D. H. LONGSTREET (EDITOR), *Software Maintenance and Computers,* IEEE Computer Society Press, Los Alamitos, CA, 1990.

[Loukides and Oram, 1997] M. K. LOUKIDES AND A. ORAM, *Programming with GNU Software,* O'Reilly and Associates, Sebastopol, CA, 1997.

[Mancl and Havanas, 1990] D. MANCL AND W. HAVANAS, "A Study of the Impact of C++ on Software Maintenance," *Proceedings of the IEEE Conference on Software Maintenance,* San Diego, CA, November 1990, pp. 63–69.

[Meyer, 1990] B. MEYER, "Lessons from the Design of the Eiffel Libraries," *Communications of the ACM* **33** (September 1990), pp. 68–88.

[Onoma, Tsai, Poonawala, and Suganuma, 1998] A. K. ONOMA, W.-T. TSAI, M. H. POONAWALA, AND H. SUGANUMA, "Regression Testing in an Industrial Environment," *Communications of the ACM* **42** (May 1998), pp. 81–86.

[Parnas, 1999] D. L. PARNAS, "Ten Myths about Y2K Inspections," *Communications of the ACM* **42** (May 1999), p. 128.

[Pigoski, 1996] T. M. PIGOSKI, *Practical Software Maintenance: Best Practices for Managing Your Software Investment,* John Wiley and Sons, New York, 1996.

[Rochkind, 1975] M. J. ROCHKIND, "The Source Code Control System," *IEEE Transactions on Software Engineering* **SE-1** (October 1975), pp. 255–65.

[Rosenblum and Weyuker, 1997] D. S. ROSENBLUM AND E. J. WEYUKER, "Using Coverage Information to Predict the Cost-Effectiveness of Regression Testing Strategies," *IEEE Transactions on Software Engineering* **23** (March 1997), pp. 146–56.

[Rothermel and Harrold, 1996] G. ROTHERMEL AND M. J. HARROLD, "Analyzing Regression Test Selection Techniques," *IEEE Transactions on Software Engineering* **22** (August 1996), pp. 529–51.

[Rugaber and White, 1998] S. RUGABER AND J. WHITE, "Restoring a Legacy: Lessons Learned," *IEEE Software* **15** (July/August 1998), pp. 28–33.

[Sneed, 1995] H. M. SNEEED, "Planning the Reengineering of Legacy Systems," *IEEE Software* **12** (January 1995), pp. 24–34.

[Teng, Jeong, and Grover, 1998] J. T. C. TENG, S. R. JEONG, AND V. GROVER, "Profiling Successful Reengineering Projects," *Communications of the ACM* **41** (June 1999), pp. 96–102.

[Tichy, 1985] W. F. TICHY, "RCS—A System for Version Control," *Software—Practice and Experience* **15** (July 1985), pp. 637–54.

[von Mayrhauser and Vana, 1997] A. VON MAYRHAUSER AND A. M. VANA, "Identification of Dynamic Comprehension Processes during Large Scale Maintenance," *IEEE Transactions on Software Engineering* **22** (June 1996), pp. 424–37.

[Wilde, Matthews, and Huitt, 1993] N. WILDE, P. MATTHEWS, AND R. HUITT, "Maintaining Object-Oriented Software," *IEEE Software* **10** (January 1993), pp. 75–80.

부록 1. 브로드랜즈지역 아동병원

브로드랜즈지역 아동병원(*Broadlands Area Children's Hospital*; BACH)은 브로드랜즈시의 반경 926km안에서 살고 있는 어린이들에게 전문화된 소아과치료를 제공하여준다. 이밖에 세계 각지에서 환자들이 자기 나라에서는 받을수 없는 치료봉사를 받기위하여 브로드랜즈로 온다. 입원환자들의 평균입원기간은 3.6일이다. 그러나 일부 어린이들은 병을 회복하는데 상당히 오랜 시일이 걸리며 6개월부터 9개월까지 입원하는 경우가 드물지 않다.

리상적인 경우에 모든 환자들은 BACH에 입원해 있는 전 기간 보호자와 함께 있게 된다. 그러나 여러가지 리유로 인하여 일부 어린이들은 입원 전 기간 또는 일정한 기간 자기 혼자 있게 된다. 다른 한가지 문제는 BACH환자들의 대다수 부모들이 재정적곤난을 겪고 있는것이다. 부모들은 자기 아이들에게 놀이감, 학습장, 크레용 등을 즐겨 보장해주어야 하겠지만 그들에게는 그렇게 할 재정적여유가 없을수 있다.

BACH의 어린이부모대리기업체(*Child-Parent Extension*; CPE)는 부모들의 요구를 충족시킬수 있는 그 무엇이나 다하는 자원자들로 조직된 기업체이다. 부모들이 함께 있지 못하게 되는 어린이들이나 부모들이 재정적곤난을 겪고 있는 어린이들은 이 기업체의 특별한 관심사로 된다. 부모들의 요구는 CPE의 3개의 분과위원회에 의해 충족되게 된다. 즉 매 어린이를 매일 적어도 30min동안 누군가가 면회하도록 하는 임무를 수행하는 꼬마친구분과(*Wee Friends*; WF), 부모들이 자기 아이들을 면회할수 있도록 무임수송 및 숙박을 보장하여 주는 가족면회보장분과(*Joining Children with their Families*; JCF), 부모들의 수입이상으로 어린이들에게 놀이감, 학습장, 기타 선물들을 보장하여 주는 임무를 수행하는 아동위문분과(*Juvenile Comforts*; JC)가 있다(이 숙어들에 대한 정보는 다음의 《알고 싶은 문제》에서 보시오.). 일부 경우에 이 분과위원회들은 서로 협동한다. 실례로 부모들은 JCF의 도움으로 병원에 와서 JC로부터 자기 아이들에게 줄 선물들을 받게 된다. 이 분과위원회들은 모두 최대의 비밀성을 준수하고 있다. 즉 환자나 부모들은 이러한 자선사업을 누가 하였는가를 전혀 알수가 없다. 어린이부모대리기업체는 병원의 콤퓨터체계를 리용하여 자기들의 원조를 필요로 하는 어린이들을 계속 추적하고 싶어 하지만 병원측은 의학적비밀준수를 리유로 이것을 허용하지 않을것이다. 이로부터 CPE는 자기 자신의 쏘프트웨어제품을 만들기로 결심하였다.

이름들은 모두 다음과 같은 형식을 취한다(아래에서 선택마당들을 중괄호안에 넣었다.).

세례명(15개 문자까지)
[성의 첫 문자(1개 문자)]
이름(15개 문자까지)
[이름뒤붙이(5개 문자까지)]

알고 싶은 문제

부록 1의 략어와 관련하여 유모아적으로 말하면 요한 쎄바스치얀 바흐(Johann S bastian Bach; BACH)(역자주 : 독일의 작곡가(1685-1750))는 자기의 첫 안해인 사촌 마리아 Maria) 한테서 7명의 자식을 보았고 두번째 안해인 안나 마그달레나(Anna Magdalena)에게 13명의 자식을 더 보았다. 이 아이들은 모두 상당한 음악적재능을 보여 주었다. 그중 네 아들들은 응당 유명한 작곡가로 된것으로 생각된다. 즉 칼 필리프 엠마뉴엘(Carl phillipp E anuel; CPE), 윌헬름 프리에데만(Wilhelm Friedemann; ER), 요한 크리스토프 프레드리히 Joham Chistoph Friedrich; JCF)와 요한 크리스치얀 (Johann Christian; JC)들이다.

주소들은 모두 다음과 같은 형식을 취한다.

주소의 첫행 (25개 문자까지)
[주소의 두번째 행(25개 문자까지)]
도시(14개 문자까지)
나라, 주 혹은 지역(14개 문자까지)
[우편대호(10개까지의 문자, 수자, 련결부호, 공백)]
고향(20개 문자까지)

전화번호들은 모두 다음의 형식을 취한다.

[+기호와 뒤이어 나라코드(3개의 수자까지)와 련결기호(-)]
지역코드(3개의 수자까지)
련결기호(-)
지역코드(4개의 수자까지)
련결기호(-)
수자(4개의 수자까지)
[Ext에 뒤이어 6개수자까지의 확장수자]

날자들은 모두 다음의 형식을 취한다 :

YYYY/MM/DD

시간은 모두 다음의 형식을 취한다 :

24h 표시법에 의한 HH:MM

모든 재정자료들은 다음과 같은 형식을 가진다.

999,999,999,99딸라까지

매 환자에 대하여 다음의 정보를 기억한다.

환자이름
환자번호 (3개의 수자, 련결부호, 2개의 수자, 련결부호, 4개의 수자)
환자부모 혹은 보호자 1 :
이름

환자와의 관계(어머니, 아버지, 이붓어머니, 이붓아버지, 기타)
집주소
[지역주소]
집전화번호
[지역전화번호]

[환자부모 혹은 보호자 2 :
(우와 같다.)]

환자위치 :
두자리수자의 층번호, 련결부호, 세자리수자의 방번호

담당의사 :
이름
병원경보기수자(네자리수자)

담당간호원
(우와 같다.)

입원날자
퇴원날자
부모 혹은 보호자가 브로드랜즈지역 아동병원의 926km 반경이내에서 살고 있는가 (y/n)?
부모 혹은 보호자가 확실히 JCF의 방조를 요구하는가 (y/n)?
부모 혹은 보호자가 확실히 WF의 방조를 요구하는가 (y/n)?
부모 혹은 보호자가 확실히 JC의 방조를 요구하는가 (y/n)?
가족면회보장분과경력(후에 보시오.)
꼬마친구분과경력(후에 보시오.)
아동면회분과경력(후에 보시오.)

제품이 개발되고 있는 기간에는 퇴원환자기록들을 장기기억에 옮길 필요가 없다.

WF에 그들의 봉사가 필요한가를 통보하는것은 담당간호원의 임무이다. 어떤 환자가 보호자와 함께 있지 않다는것(또는 부모가 집으로 돌아 갔다는것)을 관찰한 간호원은 그 환자의 기록을 갱신한다. 요청을 받을 때 WF사무소에 WF방조를 요구하는 환자의 이름, 번호, 방번호에 대한 목록이 제공되여야 한다.

WF는 자원면회자명단을 보관하고 있다. 매 자원자들에 대하여 다음과 같은 정보를 기억하여야 한다 :

방문자이름
방문자번호(환자번호와 같은 형식)
집주소
집전화번호
[사무실전화번호]
[확스번호]

[경보기번호]

주간 매일 아침, 점심마다 자원자가 면회하려는 환자의 최대수(1～4 : 0은 《그 시각에 리용할수 없다는것》을 의미한다.)

면회자가 환자를 면회한 다음에 다음의 정보가 입력되여야 한다.

면회자번호
환자번호
면회날자
면회시작시간
면회끝시간
면회자는 이 환자를 또 면회하려고 하는가 (y/n)?
설명문(100개 문자이내의 본문)

매일 오후 WF사무소는 다음날 병문안 오리라고 예상되는 자원자들의 명단, 그들이 면회하려는 최대환자수, 자원자들의 전화번호와 관련된 정보를 요구한다. 이 제품은 매 환자마다 한명의 자원자를 할당한다. WF프로그람의 대중성으로 인하여 어떤 주어 진 날에 면회자가 너무 적어 질 문제는 없다. 반대로 자원자들을 유지하기 위하여 제품은 시간표가 작성된 매 자원자들에게 적어도 한명의 환자가 할당되도록 하여야 한다. 그러나 일정작성알고리듬은 이미 어떤 환자를 면회하고 그 환자를 또 면회하기를 바라는 면회자에게 우선권을 주어야 한다. 이밖에 계획된 어떤 면회자가 오지 않는 경우에 대처하여 두명의 면회자를 《대리》당번으로 할당하여야 한다.

면회자들이 할당되였을 때 제품은 환자들의 이름, 번호, 방번호를 렬거하여 주는 매 면회자를 위한 인쇄지를 인쇄한다. 매 면회자들의 인쇄지에 대한 정보를 병합하고 있는 주목록표를 인쇄한다.

어떤 보호자가 어떤 어린이를 면회하기 위한 재정적방조를 필요로 한다는것을 담당간호원이 알아 차린 경우에 그것을 가족면회보장분과에 통보하는것도 담당간호원의 임무이다. 가족들이 이와 같은 봉사를 요구한다고 간주될 때 련관된 어린이의 이름이 그 목록의 아래에 추가된다. 그러면 받을수 있는 자금이 그 목록의 상단에 있는 어린이의 가족에 할당된다. 그다음 그 어린이는 목록의 하단으로 이동하게 된다(예산의 제약으로 인하여 JCF는 한명의 면회자에게만 지불할것이다. 또한 부모들이 브로드랜즈의 926km 반경밖에서 살고 있는 어린이에 대해서는 JCF가 지불을 하지 않는다.).

하나의 기록에 매 가족면회에 대한 내용을 보관한다 :
환자번호
환자이름
부모 또는 보호자의 이름
환자와의 관계(앞을 참고하시오.)
면회시작날자
면회끝날자

면회를 위한 예산량
실지 면회비용
설명문(100개 문자이내의 본문)

마지막으로 담당간호원이 입력한 정보에 기초하여 가족위문분과사무소를 위한 요청에 대한 목록이 인쇄된다. 이 절차는 JCF에서 리용한 목록과 류사하다. 즉 어린이가 이러한 봉사를 요구한다고 간주될 때 련관된 어린이가 목록의 하단에 추가된다. 그리고 받을수 있는 자금이 목록의 상단에 있는 어린이에게 할당된다. 그다음 그 어린이는 목록의 하단으로 이동하게 된다. 그러나 어디에 살고 있는가에 관계없이 봉사를 요구하는 모든 어린이들이 JC의 적격자로 된다. 어떤 어린이들을 면회하는 어떤 보호자가 그 어린이에게 줄 선물을 가지고 있도록 JC와 JCF를 통합하는것이 필요하다. 그러므로 어떤 JCF면회가 JC의 적격자로 되는 어떤 어린이에게 대하여 계획될 때 그 어린이는 JC목록의 상단으로 이동하여야 한다. JCF로부터 JC에로 이행에 관한 보고서는 다음의 내용을 포함한다.

환자번호
환자이름
면회시작날자
설명문(100개 문자이내)
하나의 기록이 매 선물에 대한 내용을 보관한다.
환자번호
환자이름
선물을 받은 날자
선물내용(50개 문자이내)
선물비용
설명문(100개 문자이내)

최종제품이 적당할 때 병원측은 어떤 환자가 입원할 때마다 다음과 같은 정보 즉 환자의 이름, 번호, 부모 또는 보호자, 주소, 담당의사, 담당간호원자료를 제공할것이다. 제품이 개발되는 기간 이 정보는 어떤 환자가 CPE봉사를 요구할 때마다 BACH직원에 의해 수동적으로 입력될것이다.

부록 2. 쏘프트웨어공학자원

쏘프트웨어공학주제에 대한 정보를 더 얻는데는 두가지 방법이 있다 . 즉 잡지들과 대회토론회회보를 리용하는 방법이 있고 다음에는 인터네트와 World Wide Web를 리용하는 방법이 있다.

*IEEE Transactions on Software Engineering*와 같은 쏘프트웨어공학전용잡지들이 있다. 또한 *Communications of the ACM*과 같은 보다 일반적인 잡지들이 있는데 여기에는 쏘프트웨어공학에 대한 중요한 기사들이 게재되여 있다. 지면상리유로 하여 아래에 두 부류의 잡지들만 선택하여 제시하였다.

ACM Computing Reviews
ACM Computing Surveys
ACM SIGCHI Bulletin
ACM SIGPLAN Notices
ACM SIGSOFT Software Engineering Notes
ACM Transactions on Computer Systems
ACM Transactions on Programming Languages and System
ACM Transactions on Software Engineering and Methodology
Communications of the ACM
Computer Journal
IBM Journal of Research and Development
IBM System Journal
IEEE Computer
IEEE Software
IEEE Transactions on Software Engineering
Journal of Software Maintenance: Research and Practice
Journal of Systems and Software
Software Engineering Journal
Software─Practice and Experience

이밖에 여러 대회회보에는 쏘프트웨어공학에 대한 중요한 론문들을 포함하고 있다. 또한 그 주제들에 대하여 아래에 제시해 주었다. 대부분의 대회들에는 그 략칭이나 발기기업체의 이름을 괄호안에 주었다.

ACM SIGPLAN Annual Conference (SIGLAN)
ACM SIGSOFT Symposium on the Foundations of Software Engineering(FSE)
Conference on Human Factors in Computing Systems(CHI)
Conference on Object-Oriented Programming System, Languages, and Applications(OOPSLA)
International Computer Software and Applications Conference(COMPSAC)
International Conference on Software Engineering(ICSE)
International Conference on Software Maintenance(ICSM)
International Conference on Software Reuse(ICSR)
International Conference on the Software Process(ICSP)
International Software Architecture Workshop(ISAW)
International Symposium on Software Testing and Analysis(ISSTA)
International Workshop on Software Configuration Management(SCM)
International Workshop on Software Specification and Design(IWSSD)

인터네트는 쏘프트웨어공학에 대한 또 하나의 가치 있는 정보원천으로 되고 있다. Usenet news group과 관련하여 다음의 두개가 시종일관하게 유용한것으로 되고 있다.

comp.object
comp.software-eng

관련된다고 보는 항목들을 때때로 포함하고 있는 기타 다른 newsgroup에는 다음과 같은것들이 있다.

comp.lang.c++. moderated
comp.lang.java.programmer
comp.risks
comp.software.config-mgmt

부록 3. 항공음식전문회사 실례연구 : C 신속원형

항공음식전문회사 실례연구를 위한 C신속원형은 WWW의 www.mhhe.com/engcs/compsci/schach에서 리용할수 있다.

부록 4. 항공음식전문회사 실례연구 : JAVA 신속원형

항공음식전문회사 실례연구를 위한 Java신속원형은 WWW의 www.mhhe.com/engcs/compsci/schach에서 리용할수 있다.

부록 5. 항공음식전문회사 실례연구 : 구조화체계분석

1단계 : 자료흐름선도를 작성한다.
　　이것은 11.14에서 서술한다.
2단계 : 어느 부분을 어떻게 콤퓨터화하겠는가를 결정한다.
　　DFD에 보여 준것처럼 완전한 제품을 건반구동형식의 직결독립형제품으로 콤퓨터화한다.
3단계 : 자료흐름들의 세부들을 작성한다.
　　특별식사류형(어린이용, 당뇨병환자용, 유태인용, 회교도용, 무유당, 저카로리, 저콜레스테롤, 저지방, 저단백, 저염, 해산물, 남새료리)
　　승객항목
　　예약식별자(6개의 대문자)
　　승객이름
　　　세례명(15개 문자까지)
　　　[성의 첫 문자(1개 문자)]

이름(15개 문자까지)
[이름뒤붙이(5개 문자까지)]
승객주소
주소의 첫행(25개 문자까지 허용)
[주소의 두번째 행(25개 문자까지 허용)]
도시(14개 문자까지 허용)
[나라, 주, 또는 지역(14개 문자까지 허용)]
[우편대호(10개 문자까지 허용되며 수자, 련결기호, 공백을 리용)]
고향(20개 문자까지 허용)
날자항목(9개 문자)
일(2자리수자)
월(3개의 대문자)
년(4자리수자)
비행수자항목(3자리수자, 빈 자리는 오른쪽에서부터 0으로 채워 진다.)
좌석번호항목
렬번호(3자리수자, 빈 자리는 오른쪽에서부터 0으로 채워 진다.)
좌석식별자(대문자로)
일반예약항목
예약식별자(6개의 대문자)
비행번호(앞의것을 보시오)
비행날자(앞의것을 보시오)
좌석번호(앞의것을 보시오)
승객항목(앞의것을 보시오)
[특별식사류형(앞의것을 보시오)]
예약항목
예약식별자(6개의 대문자)
비행번호(앞의것을 보시오)
비행날자(앞의것을 보시오)
좌석번호(앞의것을 보시오)
승객항목(앞의것을 보시오)
특별식사류형(앞의것을 보시오)
승객이 비행기에 올랐는가?(1개 문자)
식사가 오른 상태(1개 문자)
평가된 식사의 질(1~5)
특별식사항목(예약항목과 마찬가지로, 앞의것을 보시오.)
료리조달자보고항목
비행번호(앞의것을 보시오.)
비행날자(앞의것을 보시오.)
특별식사류형(앞의것을 보시오.)

오른 식사항목
    특별식사세부(앞의것을 보시오.)
    식사가 오른 상태(1개 문자)
퍼센트보고항목
    규정된대로 오른 특별식사비률(3+2개의 수자)
    특별식사를 주문한 승객이 비행기에 오른 비률(3+2개의 수자)
    특별식사를 주문하였지만 그 식사가 오르지 않은 승객들의 비률(3+2개의 수자)
오르지 않은 식사보고항목
    승객항목(앞의것을 보시오.)
    비행날자(앞의것을 보시오.)
    특별식사류형(앞의것을 보시오.)
저품질식사보고항목
    승객항목(앞의것을 보시오.)
    비행날자(앞의것을 보시오.)
    특별식사류형(앞의것을 보시오.)
    평가된 식사의 질(1~5)
저염식사보고항목
    비행번호(앞의것을 보시오.)
    비행날자(앞의것을 보시오.)
    평가된 식사의 질(1~5)
시작 및 마감날자
    시작날자(앞의것을 보시오.)
    마감날자(앞의것을 보시오.)
승객식사평가
    예약식별자(6개의 대문자)
    평가된 식사의 질(1~5)

4단계 : 공정의 론리를 정의한다.
예약을 입력한다.
    승객으로부터 예약세부자료를 얻어 낸다.
오른 식사보고서를 작성하고 조사한다.
    비행기안내원들은 모든 식사가 다 올랐다는것을 표식한다. 대응하는 식사탑승상태마당이 후에 Y로 설정된다.
탑승보고서를 작성한다.
    어떤 특정한 비행번호에 대한 특별식사요구들이 인쇄된다.
특별식사요구를 추출한다,
    만일 예약에 어떤 특별식사가 포함되면 그 예약은 적당히 표식한다.

우편엽서를 생성한다.

어떤 특별식사를 받은 매 승객에 대하여 그 승객의 세부자료가 들어있는 우편엽서를 생성한다.

우편엽서를 조사한다.

우편엽서가 되돌아 올 때 예약식별자를 확인하고 평가된 식사품질을 기록한다.

료리조달자보고서를 작성한다.

어떤 주어 진 비행날자와 비행기번호에 대하여 요청된 식사를 인쇄한다.

퍼센트보고서를 작성한다.

어떤 주어 진 시작날자와 마감날자에 대하여 규정된대로 오른 특별식사의 퍼센트, 특별식사를 주문한 승객이 비행기에 탑승한 퍼센트, 특별식사를 주문하였지만 오르지 않은 비행기에 탑승한 승객들의 퍼센트를 계산하고 인쇄한다.

오르지 않은 식사보고서를 작성한다.

어떤 주어 진 시작날자와 마감날자에 대하여 식사가 한번이상 오르지 않은 매 승객의 이름과 주소 및 그 사건의 발생날자를 인쇄된다.

저품질보고서를 작성한다.

어떤 주어 진 시작날자와 마감날자에 대하여 평가한 식사품질이 4 또는 그이하인 승객들의 이름과 주소 및 그 사건의 발생날자, 식사류형을 인쇄한다.

저염식사보고서를 작성한다.

어떤 주어 진 시작날자와 마감날자, 봉사한 매 저녁식사에 대하여 비행기번호와 그 사건의 출현날자, 평가된 식사품질을 인쇄한다.

5단계 :자료저장을 정의한다.

RESERVATION DATA(187바이트)

예약식별자(6개의 대문자)

비행번호(3자리수자, 빈 자리는 오른쪽에서부터 0으로 채워 진다.)

비행날자(9자리수자:2자리수자로 된 날자, 3개의 대문자로 된 월, 4자리수자로 된 년)

좌석번호(3자리수자, 빈 자리는 오른쪽에서부터 0으로 채워 진다.)

예약식별자(6개의 대문자)

세례명(15개 문자까지)

성의 첫 문자(1 개 문자)

이름(15개 문자까지)

이름뒤붙이(5개 문자까지)

주소의 첫행(25개 문자까지 허용)

주소의 두번째 행(25개 문자까지 허용)

도시(14개 문자까지 허용)

나라, 주, 또는 지역(14개 문자까지 허용)

우편대호(10개 문자까지 허용되며 수자, 련결기호, 공백을 리용)
고향(20개 문자까지 허용)
특별식사류형(15개 문자까지)
특별식사표식(1개 문자)
SPECIAL MEALS DATA(192바이트)
RESERVATION DATA(앞의것을 보시오.)
승객이 비행기에 올랐는가?(1개 문자)
식사가 오른 상태(1개 문자)
평가된 식사의 질(1~5)
RESERVATION DATA에 대한 즉시접근은 예약식별자(1차)와 이름과 좌석번호에 의해 실현된다.
SPECIAL MEALSS DATA에 대한 즉시접근은 예약식별자(1차)와 이름, 좌석번호, 식사류형에 의해 실현된다.

6단계: 물리적자원들을 정의한다.

RESERVATION DATA
색인된 순서파일
1차색인 예약식별자
2차색인 이름과 좌석번호
SPECIAL MEALS DATA
색인된 순서파일
1차색인 예약자료
2차색인 이름과 좌석번호, 식사류형

7단계: 입력/출력명세서들을 결정한다.

다음의 공정들을 위한 입력스크린이 설계될것이다.
예약을 입력한다.
일반적인 예약항목을 입력한다.
료리조달자보고서를 작성한다.
비행번호와 비행날자를 입력한다.
퍼센트보고서를 작성한다.
시작날자와 마감날자를 입력한다.
실어 지지 않은 식사보고서를 작성한다.
시작날자와 마감날자를 입력한다.
저품질보고서를 작성한다.
시작날자와 마감날자를 입력한다.
저염보고서를 작성한다.
시작날자와 마감날자를 입력한다.

다음의 보고서들을 인쇄할것이다：
탑승보고서

승객이름, 좌석번호, 식사류형

우편엽서

승객이름, 비행날자, 비행번호

료리조달자보고서

비행날자, 비행번호, 식사류형

퍼센트보고서

시작날자, 마감날자, 명시되여 있는대로 오른 특별식사의 비률, 특별식사를 주문한 비행기에 오른 승객들의 비률, 특별식사를 주문했으나 그 식사가 오르지 않은 비행기에 오른 승객들의 비률

오르지 않은 식사보고서

시작날자, 마감날자, 승객이름, 승객주소, 발생날자

저품질보고서

시작날자, 마감날자, 승객이름, 승객주소, 식사류형, 평가된 식사의 질, 발생날자,

저염식사보고서

시작날자, 마감날자, 비행번호, 비행날자, 평가된 식사의 질

다음의 항목들이 조사될것이다.

탑승보고서조사

예약식별자, 식사가 오른 상태

우편엽서조사

예약식별자, 평가된 식사의 질

8단계 : 규격화를 수행한다.

두개의 완성된 예약기록들은 한개의 512바이트의 디스크블로크에 채울수 있다. 만일 비행당 128건의 예약이 있고 항공음식전문회사가 봉사하는 64개의 매 비행장들로부터 하루에 8번의 비행이 있다고 가정하면 128GB의 직결레코드가 존재하게 된다.

매일 512개의 탑승보고서를 조사해야 할것이다. 만일 승객의 5%가 특별식사를 요청하며 그중 절반이 자기들의 우편엽서를 되돌려 보낸다면 매일 1,500개의 우편엽서를 조사해야 할것이다.

9단계 : 하드웨어요구사항을 결정한다.

매 비행장관문에 2대의 말단콤퓨터와 매 탑승수속지점에 한대의 말단기가 요구된다. 이 말단들은 항공음식전문회사 망과 련결될것이다. 만일 항공음식전문회사가 자기가 봉사하는 64개의 매 비행장에 두개의 관문을 가지고 있으며 비행장마다 8개의 추가적인 탑승수속말단기가 요구된다면 64개의 말단기를 구입해야 할 필요가 있다.

현재 매 비행기에 탑승된 인쇄기들은 탑승보고서를 찍는데 충분할것이다. 본사에 있는 인쇄기들은 인쇄해야 할 각종 보고서들은 물론 매일 3,000개의 우편엽서를 찍는데 충분할것이다.

매일 512개의 탑승보고서와 1,500개의 우편엽서를 조사할수 있는 조사기를

구입해야 한다. 관리자측은 모든 조사를 본사에서 수행할것인가(이 경우에 한대의 조사기와 한대의 여벌조사기가 적당하다.) 아니면 조사를 여러 싸이트에서 수행하겠는가를 결정하여야 한다.

부록 6. 항공음식전문회사 실례연구 : 쏘프트웨어프로젝트관리계획

여기서 제시하는 해결방안은 3명 즉 회사 사장 레스(Les)와 두명의 쏘프트웨어공학자들로 구성된 작은 쏘프트웨어개발기업체가 항공음식전문회사제품을 개발할 때의 쏘프트웨어프로젝트관리계획(*Software Project Management Plan*; SPMP)이다.

1. 서 론

1.1 프로젝트개괄 이 프로젝트의 목적은 특별식사를 요청하는 승객들에게 그것을 계속 공급하겠는가를 결정할 때에 항공음식전문회사를 지원하는 쏘프트웨어제품을 개발하는것이다. 이 제품은 의뢰자가 어떤 예약을 입력하고 어떤 승객의 탑승수속을 하며 특별식사계획을 관리하기 위해 필요한 정보를 발생할수 있게 한다. 제품은 매 비행에서의 특별식사요구, 식사품질, 오르지 않은 식사와 관련한 보고서를 생성할것이다.

시간, 예산, 개인적요구사항들은 다음과 같다.
요구사항확정단계(1주일, 두명의 개발팀성원, 1,750딸라)
객체지향분석단계(1주일, 두명의 개발팀성원, 1,750딸라)
계획단계(1주일, 두명의 개발팀성원, 1,750딸라)
설계단계(2주일, 두명의 개발팀성원, 3,500딸라)
실현단계(3주일, 3명의 개발팀성원, 7,875딸라)
통합단계(2주일, 3명의 개발팀성원, 5,250딸라)
실현단계와 통합단계는 제15장에서 서술한바와 같이 결합된다.
총 개발기일은 10주이며 총 내부비용은 21,875딸라이다.

1.2. 프로젝트배포물 완성된 원천코드는 사용자지도서와 조작지도서와 함께 프로젝트에 착수한후 10주후에 배포될것이다. 의뢰자는 제품이 배포될 때까지 권고된 하드웨어와 체계쏘프트웨어를 구입할 책임을 지게 될것이다.

1.3. SPMP의 진화 SPMP에서의 모든 변경은 그것을 실현하기전에 사장의 동의를 받아야 한다. 모든 변경들은 SPMP가 정정되고 갱신될수 있도록 문서로 작성하여야 한다.

1.4. 참고자료 회사의 하나의 코드작성표준, 문서작성기준, 시험기준

1.5. 정의와 략어 SPMP－Software Project Management Plan(쏘프트웨어프로젝트관리

계획)

2. 프로젝트의 구성

2.1. 공정모형 리용하는 쏘프트웨어생명주기모형은 신속원형을 가진 폭포모형이다. 명세서는 다나(Dana)와 린드씨(Lindsey)가 작성하였으며 레스(Les)와 의뢰자의 면담에서 의뢰자에 의해서 검증되였다. 사장은 전반적인 설계를 검토한다. 코드작성은 다나와 린드씨가 진행하였다. 그들은 상대방의 코드를 시험하며 사장은 통합시험을 지휘한다. 그다음 확장시험은 3명모두에 의해서 진행된다. 이 모든 활동에 필요한 시간은 서론에 제시된다.

2.2. 기업체의 구성 개발팀은 레스(사장), 다나와 린드씨(쏘프트웨어기사)로 구성된다.

2.3. 기업체의 경계와 호상결합 이 프로젝트상에서의 모든 작업은 레스, 다나, 린드씨에 의해서 수행된다. 레스는 매주 의뢰자와 만나 프로젝트진척정형을 보고하고 가능한 변화와 변경들에 대하여 토론한다. 일정표 또는 예산에 영향을 줄 모든 중요한 변경들에 대해서는 사장의 허가를 받고 문서화되여야 한다. 외부로부터 그 어떤 쏘프트웨어품질보증성원도 인입시키지 않는다. 매 사람이 다른 사람의 작업제품을 시험하여 개발자가 아닌 다른 사람이 그 시험을 하여 얻게 되는 리득을 획득하게 한다.

2.4. 프로젝트책임 매 성원들은 자기가 코드작성한 모듈의 질에 대하여 책임진다. 사장은 클라스정의표와 보고서모듈들을 다루며 다나는 비행정보를 처리하는 모듈들을 작성하며 린드씨는 승객정보를 처리하는 모듈들을 코드작성한다. 사장은 제품의 모듈통합과 전반적품질이 의뢰자의 요구를 충족시키는가를 감독한다.

3. 관 리 공 정

3.1. 관리목적과 우선권 최종목적은 오유 없는 제품을 제때에 제한된 예산안에서 배포하는것이다. 만일 이 목적을 달성할수 없다면 매 비행에 대한 특별식사를 주문하는데 요구되는 루틴들을 완성하는데 보다 높은 우선권이 부여되며 기타 보고서들을 보다 낮은 우선권을 가지게 된다.

3명의 개발팀성원들은 자기들이 맡은 모듈상에서 개별적으로 작업한다. 레스의 역할은 다른 두 사람의 매일 작업진척정형을 료해하고 통합을 료해하여 제품의 전반적질을 책임지며 의뢰자와의 호상련계를 맺는것이다. 개발팀성원들은 매일 작업마감에 만나서 문제점들과 진척정형에 대하여 토론한다. 의뢰자와의 공식면담은 매주 말에 진행되며 이때 프로젝트진척정형을 보고하고 어떤 변경이 만들어 질 필요가 있겠는가를 결정한다. 사장은 일정과 예산요구조건들이 만족된다는것을 담보하게 된다. 위험요소관리 역시 사장의 책임이다.

오유를 최소로 하고 사용자와의 친밀성을 최대로 보장하는것이 사장의 최우선책임이다. 사장은 또한 모든 문서들에 대하여 책임지며 그것이 갱신된다는것을 담보하여야 한다.

3.2. 가정, 종속, 계약 인수판정기준이 명세서에 렬거된다. 이밖에

최종기한이 만족되여야 한다.

예산제약이 만족되여야 한다.

제품은 신뢰성이 있어야 한다.
추가적인 모듈들이 후에 추가될수 있도록 구성은 열린 방식으로 되여야 한다.
제품은 의뢰자의 하드웨어조건을 만족시켜야 한다.
제품은 사용자에게 친절하게 만들어 져야 한다.

3.3. 위험요소관리 위험인자들과 그것을 추적하기 위한 기구는 다음과 같다.

○ 새 제품과 비교할수 있는 현존 쏘프트웨어는 없다. 따라서 현존 제품과 병렬로 실행될수 없다. 그러므로 제품은 확장시험을 받아야 한다.
○ 의뢰자는 콤퓨터에 대한 경험이 없다고 가정된다. 그러므로 명세작성단계와 의뢰자와의 정보교환에 특별한 주의를 돌려야 한다. 제품은 될수록 사용자에게 친절하도록 개발되여야 한다.
○ 기본설계오유가 언제나 존재할수 있다. 그러므로 확장시험은 설계단계에서 수정된다. 또한 매 개발팀성원들이 자기가 작성한 코드를 우선 시험하고 다른 성원의 코드를 시험한다. 사장은 통합시험을 책임진다.
○ 제품은 규정된 기억요구들과 응답시간들을 충족시켜야 한다. 이것은 제품의 규모가 작기때문에 기본문제로 될것 같지는 않지만 사장은 개발 전 기간에 걸쳐 이에 대하여 료해해야 한다.
○ 드물게 하드웨어가 고장날 경우가 있다. 이 경우에는 다른 기계가 임대될것이다. 만일 콤파일러에 장애가 있다면 그것은 교체될것이다. 이것들은 하드웨어와 콤파일러공급자들에게서 받게 되는 담보서에 포함된다.

3.4. 검사 및 조종기구 사장은 모든 검토와 재정검사를 책임진다. 이것은 사장이 매일 개발된 성원들과 면담하는것을 통해 실현된다. 매 면담에서 다나와 린드는 그날 작업진척정형과 문제점들을 보고한다. 사장은 작업이 예상대로 진척되고 있는가 그리고 명세서와 SPMP를 준수하고 있는가를 결정한다. 개발팀성원들이 직면하고 있는 중요한 문제점들은 즉시 사장에게 보고된다.

3.5. 직원채용계획 레스는 10주 전 기간 필요하며 첫 5주간은 경영자적권한만을 행사하며 나머지 5주간은 경영자와 프로그람작성자의 역할을 수행한다. 다나와 린드는 10주 전 기간 필요하다.

4. 기술적공정

4.1. 방법, 도구, 기법 신속원형을 가진 폭포모형이 리용된다. 신속원형은 C로 작성된다. 명세서는 객체지향분석수법을 리용하여 작성된다. 객체지향설계가 리용된다. 원천코드는 C++ 또는 Java로 작성되며 개인콤퓨터상의 Linux환경하에서 실행된다. 문서작성과 코드작성은 회사의 표준에 따라 수행된다.

4.2. 쏘프트웨어문서 쏘프트웨어문서는 회사의 표준에 준하여 작성된다. 문서에 대한 검토는 공정모형의 매 단계가 완성될 때 레스의 지도하에 수행된다. 이것은 어떤 특정한 단계에서의 모든 문서작성이 다음단계가 시작될 때까지 완성되게 된다는것을 담보하고 있다.

4.3. 프로젝트지원기능 품질보증은 2.1에서 서술한것과 마찬가지로 수행된다.

5. 작업패키지, 일정표, 예산

5.1. 작업패키지 여기에 포함되는 항목들은 비행, 승객, 식사이다. 특별식사에 대한 정보들을 저장하고 식사요구, 식사품질, 식사가 올랐는가에 관한 보고서를 생성하기 위한 루틴들이 요구된다. 이 매개 항목들에 대한 방법들은 독립적으로 만들어 진다. 개발팀성원들은 항시적으로 정보교환상태에 있으며 이것은 클라스의 무모순성을 담보하여야 한다.

5.2. 종속성 종속성은 공정모형에 명시된것과 마찬가지이다. 특히 선행한 단계의 작업제품이 사장의 허가를 받기전까지는 그 어떤 단계도 시작될수 있다.

5.3. 자원요구 표준리눅스도구와 함께 리눅스환경하에서 동작하는 3대의 콤퓨터가 요구된다.

5.4. 예산 및 자금분배 매 단계에 대한 예산안은 다음과 같다.

요구사항확정단계	1,750딸라
객체지향분석단계	1,750딸라
계획작성단계	1,750딸라
객체지향설계단계	3,500딸라
실현단계	7,875딸라
통합단계	5,250딸라
총계	21,875딸라

5.5. 일정표

1주. 의뢰자와 만나서 요구사항들을 결정한다.
신속원형들을 생성하고 의뢰자와 사용자들이 신속원형을 허가한다.

2주. 명세서를 작성하고 명세서를 검토한다. 의뢰자가 그 문서를 허가한다.

3주. SPMP를 생성하고 SPMP를 검토한다.

4주와 5주. 객체지향설계문서를 작성하고 객체지향설계를 검토한다.
상세설계문서를 작성하고 상세설계를 검토한다.

6주～8주. 매 모듈을 실현하고 검토하며 모듈들을 시험하고 문서화한다.

9주와 10주. 모듈들을 통합하고 개별적모듈들을 검토하며 제품시험을 진행하고 문서를 검열한다.

추가적인 요구들

보안. 제품을 리용하기 위한 통과단어(암호)가 요구된다.

숙련. 숙련은 제품을 배포할 때 사장이 수행하게 된다. 이 제품은 리용하기 간단하기때문에 숙련하는데 3일이면 충분하다. 사장은 제품의 사용 첫해동안에는 무료로 발생한 문제점들에 답변을 줄것이다.

제품유지정비. 교정유지정비는 12개월이내에는 무료로 개발팀이 진행하게 된다. 능력확장과 관련한 개별적계약들이 작성된다.

부록 7. 항공음식전문회사 실례연구 : 객체지향분석

항공음식전문회사제품을 위한 완전한 객체지향분석은 12.8에 제시하였다.

부록 8. 항공음식전문회사 실례연구 : C++ 실현을 위한 설계

항공음식전문회사제품의 구성(객체지향)설계는 13.13에 제시되였다. 항공음식전문회사제품의 C++판본완전상세설계에 리용된 44개의 방법들이 클라스이름과 그 클라스이름내에서의 방법이름에 관하여 자모순서로 제시된다. 그다음에 3가지 기능들이 자모순서로 제시된다. 매 클라스의 이름은 구성방식설계에서와 같지만 문자 C로 시작된다.

모듈이름	CCatererReport::getQualifications
모듈형	방법
귀환값형	**void**
입력변수	없음
출력변수	없음
오유통보문	만일 날자가 부정확하게 입력되였으면
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CDate::validDate
해설	음식조달자로부터 비행날자와 비행번호를 검색하고 검증한다.

모듈이름	CCatererReport::print
모듈형	방법
귀환값형	**void**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CReport::print
해설	기초클라스 print방법을 확장한다 ; 즉 식사를 추적할수 있는 배렬을 초기화하고 기초클라스 print방법을 호출하며 그다음 배렬을 인쇄한다.

모듈이름	CCatererReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	그 어떤 정보를 인쇄하는것이 아니라 음식조달자가 조달해야 하는 서로 다른 종류의 식사를 추적할수 있는 배렬을 갱신한다.

모듈이름	CCatererReport::qualifiesForRecord
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	만일 사용자가 명기한대로 비행날자와 비행번호가 되여 있으면 이 보고서를 위한 기록을 수정한다.

모듈이름	CDate::breakupDate
모듈형	방법
귀환값형	**void**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	MMM/DD/YYYY형식으로 유효한 날자가 주어지면 각각의 년도와 날자의 옹근수요소와 월의 간략표기를 검색한다; 즉 검색값에 기초한 자료요소를 갱신한다.

모듈이름	CDate::daysInMonth
모듈형	방법
귀환값형	**Short**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CDate::isLeapYear
해설	주어 진 달의 날자수를 귀환한다.

모듈이름	CDate::isLeapYear
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	날자의 년도부분이 윤년인가를 결정한다.

모듈이름	CDate::validDate
모듈형	방법
귀환값형	**short**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CDate::breakupDate
해설	현재의 날자가 **MMM/DD/YYYY**형식의 유효한 날자인가를 결정한다. 여기서 **MMM**은 유효한 달의 생략이고 1 <= **DD** <= 31, 0 <= **YYYY** <= 2999; 날자가 유효하면 **TRUE**, 그렇지 않으면 **FALSE**를 귀환한다.

모듈이름	CFlightRecord::alreadyExists
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	fltRec.dat
변경된 파일	없음
호출된 모듈	CFlightRecord::read
해설	비행기록이 이미 존재하면 TRUE를 귀환한다.

모듈이름	CFlightRecord::checkFlightNum
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	비행번호가 유효하면 **TRUE**를 귀환한다(실례로 3개까지의 수자로 구성되여 있으면); 유효한 비행번호는 오른쪽에 령을 맞추어 채워 준다.

모듈이름	CFlightRecord::checkInPassenger
모듈형	방법
귀환값형	**void**
입력변수	없음
출력변수	없음
오유통보문	만일 예약ID가 6개의 문자로 되여 있지 않는 경우; 그 ID에 대한 예약이 없는경우;
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CFlightRecord::alreadyExists, getReservationID, insert
해설	승객이 특정한 예약에 대하여 기입하였는가를 확인한다.

모듈이름	CFlightRecord::checkReservationID
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	만일 예약ID가 유효하면 (실례로 6문자로 되여 있으면) **TRUE**를 귀환한다.

모듈이름	CFlightRecord::checkSeatNum
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없음
출력변수	없음
오유통보문	없음
호출된 파일	없음
변경된 파일	없음
호출된 모듈	없음
해설	만일 좌석번호가 유효하면(즉 3개이내의 수자와 그에 뒤이은 한개 문자로 구성되여 있으면) **TRUE**를 귀환한다. 유효한 좌석번호는 오른쪽으로부터 빈 자리에 령을 채워 준다.

모듈이름	CFlightRecord::getReservation
모듈형	방법
귀환값형	**void**
입력변수	없음
출력변수	없음
오유통보문	예약ID가 이미 존재하는 경우; 예약ID가 6개 문자로 구성되여 있지 않는 경우; 비행번호가 3개 수자로 되여 있지 않는 경우; 날자가 부정확하게 입력되여 있는 경우; 좌석이 이미 예약되여 있는 경우; 좌석번호가 3개이내의 수자와 그에 뒤이은 하나의 대문자로 되여 있지 않는 경우;
호출된 파일	없음
변경된 파일	없음
호출된 모듈	CDate::validDate CFlightRecord::alreadyExists, checkFlightNum, checkReservationID, checkSeatNum, insert, seatReserved CPassenger::getDescription, insert
해설	모든 승객정보를 검색한다.

모듈이름	CFlightRecord::insert
모듈형	방법
귀환값형	**void**
입력변수	없다
출력변수	없다
오유통보문	없다
호출된 파일	**fltRec.dat, tempF.dat**
변경된 파일	fltRec.dat, tempF.dat
호출된 모듈	CFlightRecord::read, write
해설	비행기록대상을 파일에 있는 적당한 위치에 삽입한다.

모듈이름	CFlightRecord::read
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	ifstream
출력변수	없다.
오유통보문	없다.
호출된 파일	입력변수에 지적된 파일
변경된 파일	없다.
호출된 모듈	없다.
해설	비행기록대상을 **fileName**으로부터 읽는다. 객체가 유효하게 읽어 지면 **TRUE**를 귀환한다.

모듈이름	CFlightRecord::scanPostcard
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	만일 예약ID가 6개 문자로 되여 있지 않는 경우; 그 ID에 대한 예약이 없는 경우;
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CFlightRecord::alreadyExists, checkReservationID, insert
해설	특정한 예약에 대하여 평가된 품질을 사용자가 입력한 값으로 설정한다.

모듈이름	CFlightRecord::scanSpecialMeals
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	비행번호가 3개 수자로 되여 있지 않는 경우; 날자가 부정확하게 입력되여 있는 경우
호출된 파일	**fltRec.dat, tempF.dat**
변경된 파일	fltRec.dat, tempF.dat
호출된 모듈	CDate::validDate CFlightRecord::checkFlightNum, insert, read, write
해설	특정한 비행에 대한 모든 예약에 대하여(예약번호+예약날자) 식사가 올랐는가를 사용자에게 질문하고 그다음에 파일을 갱신한다.

모듈이름	CFlightRecord::seatReserved
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	**fltRec.dat**
변경된 파일	없다.
호출된 모듈	CFlightRecord::read
해설	좌석번호가 이미 다른 승객에게 예약되여 있으면 **TRUE**를 귀환한다.

모듈이름	CFlightRecord::write
모듈형	방법
귀환값형	**void**
입력변수	ofstream
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	입력변수에 지적된 파일
호출된 모듈	없다.
해설	비행기록대상을 **fileName**에 써넣는다.

모듈이름	CLowSodiumReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	비행번호, 비행날자, 평가된 품질을 출력한다.

모듈이름	CLowSodiumReport::qualifiesForReport
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서에 지적된 날자범위에서 비행기에 오른 저염식사가 포함되여 있으면 이 보고서에 대하여 기록을 수정한다.

모듈이름	CNotLoadedReport::alreadyEncountered
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	ICD_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	notLoaded.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	예약ID가 이미 이 보고서에 설정되여 있는가를 결정한다(중복은 피한다.).

모듈이름	CNotLoadedReport:: markEncountered
모듈형	방법
귀환값형	**void**
입력변수	ID_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	NotLoaded.dat, tempNot.dat
변경된 파일	NotLoaded.dat, tempNot.dat
호출된 모듈	없다.
해설	현재 승객ID를 notLoaded파일에 추가한다.

모듈이름	CNotLoadedReport:: notLoadedMoreThenOnce
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	CFlightRecord::read
해설	특별식사가 한번이상 오르지 않은 승객이 있는가를 결정한다.

모듈이름	CNotLoadedReport::print
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	CFlightRecord::read CNotLoadedReport::markEncountered CPassenger::getPassenger
해설	기초클라스print방법을 호출하기전에 필요한 파일을 초기화한다.

모듈이름	CNotLoadedReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CflightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	CFlightRecord::read CNotLoadedReport::markEncountered CPassenger::getPassenger
해설	식사가 오르지 않았을 때 승객이름과 주소, 날자를 출력한다.

모듈이름	CNnotLoadedReport::qualifiesForReport
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CflightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CNotLoadedReport::alreadyEncountered, notLoadedMoreThanOnce
해설	기록이 보고서에 지적된 날자범위에 있으며 그 날자범위안에서 한번이상 식사가 오르지 않은 승객이 있을 때 이 보고서의 그 기록을 변경한다.

모듈이름	COnBoardReport::getQualifications
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	날자가 부정확하게 입력되여 있는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CDate::validDate
해설	탑승보고서에 대하여 비행날자와 비행번호를 검색하고 검증한다.

모듈이름	COnBoardReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	이 기록에 대한 승객, 좌석번호, 식사류형, 검사통을 출력한다.

모듈이름	COnBoardReport::qualifiesForReport
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	사용자가 명기한대로 비행날자와 비행번호가 되여 있으면 이 보고서에 대한 기록을 규정한다.

모듈이름	CPassenger::alreadyExists
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat
변경된 파일	없다.
호출된 모듈	CPassenger::read
해설	만일 현재 대상의 승객ID가 이미 파일에 존재하는가를 결정한다; ID가 존재하면 사용자는 파일에 보관된 값이 리용되는가를 묻는다.

모듈이름	CPassenger::getDescription
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CPassenger::alreadyExists
해설	모든 승객정보를 검사한다.

모듈이름	CPassenger::getPassenger
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	ID_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat
변경된 파일	없다.
호출된 모듈	CPassenger::read
해설	파일로부터 searchID에 등가인 passengerID를 리용하여 승객기록을 적재한다; 승객이 이미 있으면 **TRUE**를 귀환한다.

모듈이름	CPassenger::insert
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat, tempP.dat
변경된 파일	passenger.dat, tempP.dat
호출된 모듈	CPassenger::read, write
해설	파일의 적당한 곳에 승객대상을 삽입한다.

모듈이름	CPassenger::read
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	ifstream
출력변수	없다.
오유통보문	없다.
호출된 파일	입력변수에 지적된 파일
변경된 파일	없다.
호출된 파일	없다.
해설	fileName로부터 승객대상을 읽어 들인다. 객체가 유효하게 읽어 지면 **TRUE**를 귀환된다.

모듈이름	CPassenger::write
모듈형	방법
귀환값형	**void**
입력변수	ofstream
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	입력변수에 지적된 파일.
호출된 모듈	없다.
해설	fileName에 승객대상을 써넣는다.

모듈이름	CPercentageReport:print
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CReport::print
해설	기초클라스print방법을 확장한다; 즉 식사를 추적할수 있는 배렬을 초기화한다. 기초클라스print방법을 호출하고 그 배렬을 인쇄한다.

모듈이름	CPercentageReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CflightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	그 어떤 정보도 인쇄하지 않는다. 서로 다른 종류의 퍼센트에 대한 식사추적을 유지하는 여러개의 배렬을 갱신한다.

모듈이름	CPercentageReport::qualifiesForReport
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서에 지적된 날자범위에 특별식사가 있으면 이 보고서에 대한 기록을 변경한다.

모듈이름	CPoorQualityReport::printRecord
모듈형	방법
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CPassenger::getPassenger
해설	승객, 비행날자, 식사류형을 출력한다.

모듈이름	CPoorQualityReport::qualifiesForReport
모듈형	방법
귀환값형	**BOOLEAN**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서에 지적된 날자범위내에 지적된 특별식사를 주문하지 않았으면 이 보고서에 대한 기록을 변경한다.

모듈이름	CReport::getQualifications
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	날자가 부정확하게 입력되여 있는 경우 마감날자가 시작날자보다 더 앞선 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CDate::validDate
해설	수정을 위한 암묵적인 방법 즉 보고서에 정의된 시작 및 마감날자를 얻는다.

모듈이름	CReport::print
모듈형	방법
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	FltRec.dat
변경된 파일	없다.
호출된 모듈	CFlightReport::read CReport::getQualifications, printRecord qualifiesForReport
해설	보고서를 인쇄하는 암묵적인 방법; 즉 비행기록파일에 있는 모든 기록들을 반복한다. 보고서에 해당한 매 기록들에 대하여 함수 printRecord를 호출한다.

모듈이름	displayMainMenu
모듈형	**함수**
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	선택이 범위를 벗어 나는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CFlightRecord::checkInPassenger, getReservation, scanPostcard, scanSpecialMeals displayReportMenu
해설	사용자가 리용할수 있는 모든 선택을 포함한 기본차림표를 현시한다.

모듈이름	DisplayReportMenu
모듈형	**함수**
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	선택이 범위를 벗어 나는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CCatererReport::print CLowSodiumReport::print CNotLoadedReport::print* COnBoardReport::print* CPercentageReport::print CPoorQualityReport::print* *Defaults to CReport::print
해설	사용자가 리용할수 있는 모든 보고서를 포함한 차림표를 현시한다.

모듈이름	main
모듈형	**함수**
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	displayMainMenu
해설	초기화를 진행하고 displayReportMenu를 호출한다.

부록 9. 항공음식전문회사 실례연구 : JAVA실현을 위한 설계

항공음식전문회사제품의 구성방식(객체지향)설계는 13.13에 제시된다. 항공음식전문회사제품의 Java판본 완전상세설계에 리용된 39개의 방법들이 클라스이름과 그 클라스이름내에서의 방법이름에 관하여 자모순서로 제시된다. 그 다음에 3가지 기능들이 자모순서로 제시된다. 매 클라스의 이름은 구성방식설계에서와 같지만 문자 C로 시작된다.

클라스이름	AirGourmetApplication
방법이름	main
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	AirGourmetUtilities.displayMainMenu
해설	초기화를 진행하고 displayMainMenu를 호출한다.

클라스이름	AirGourmetUtilities
방법이름	displayMainMenu
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	선택이 범위를 벗어 난 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CflightRecord.checkInPassenger, getReservation, scanPostcard, scanSpecialMeals displayReportMenu
해설	사용자가 리용할수 있는 모든 선택을 포함한 기본차림표를 현시한다.

클라스이름	AirGourmetUtilities
방법이름	displayReportMenu
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	선택이 범위를 벗어난 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CCatererReport.print CLowSodiumReport.print CNotLoadedReport.print* COnBoardReport.print* CPercentageReport.print CPoorQualityReport.print* *Defaults to Creport.print
해설	사용자가 리용할수 있는 모든 보고서를 포함한 차림표를 현시한다.

클라스이름	CCatererReport
방법이름	getQualifications
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	날자가 부정확하게 입력되여 있는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	음식조달자로부터 비행날자와 비행번호를 검색하고 검증한다.

클라스이름	CCatererReport
방법이름	print
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	Creport.print
해설	기초클라스print방법을 확장한다; 식사를 추적할 수 있는 배렬을 초기화하고 기초클라스print방법을 호출하고 배렬을 인쇄한다.

클라스이름	CCatererReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CReport.print
해설	그 어떤 정보도 인쇄하지 않는다. 그러나 음식조달자가 조달해야 할 필요가 있는 서로 다른 종류의 식사를 추적할수 있는 배렬을 갱신한다.

클라스이름	CCatererReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	사용자가 명기한대로 비행날자와 비행번호가 되여 있으면 이 보고서에 대한 기록을 변경한다.

클라스이름	CFlightRecord
방법이름	alreadyExists
귀환값형	**boolean**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	비행기록이 이미 존재하면 **TRUE**를 귀환한다.

클라스이름	CFlightRecord
방법이름	checkFlightNum
귀환값형	**boolean**
입력변수	없다
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	비행번호가 유효하면(3개이내의 수자로 되여 있으면)**TRUE**를 귀환한다; 유효한 비행번호는 빈자리에 오른쪽부터 령을 채워 준다.

클라스이름	CFlightRecord
방법이름	checkFlightNum
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	예약ID가 6개 문자로 되여 있지 않는 경우 그 ID에 대한 예약이 없는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CFlightRecord.alreadyExists, getReservationID, insert
해설	특정한 예약에 대하여 승객이 기입되였는가를 확인한다.

클라스이름	CFlightRecord
방법이름	checkReservationID
귀환값형	**boolean**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	예약ID가 유효하면(6개 문자로 되여 있는 경우)**TRUE**를 귀환한다.

클라스이름	CFlightRecord
방법이름	checkSeatNum
귀환값형	**boolean**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	좌석번호가 유효하면 (3개 수자까지 되여 있는 경우) **TRUE**를 귀환한다; 유효한 좌석번호는 빈자리에 오른쪽부터 령을 채워 준다.

클라스이름	CFlightRecord
방법이름	GetReservation
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	예약ID가 이미 존재하는 경우; 예약ID가 6개 문자로 되여 있지 않는 경우; 비행번호가 3개까지의 수자로 되여 있지 않는 경우; 날자가 부정확하게 입력되여 있는 경우; 좌석이 이미 예약되여 있는 경우; 좌석번호가 3개이내의 수자와 그에 뒤이은 하나의 대문자로 되여 있지 않는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CFlightRecord.alreadyExists, checkFlightNum, checkReservationID, checkSeatNum, insert, seatReserved CPassenger.getDescription, insert
해설	모든 승객정보를 검색한다.

클라스이름	CFlightRecord
방법이름	insert
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	**fltRec.dat, tempF.dat**
변경된 파일	fltRec.dat, tempF.dat
호출된 모듈	없다.
해설	파일에 있는 특정한 곳에 비행기록대상을 삽입한다.

클라스이름	CFlightRecord
방법이름	scanPostcard
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	예약ID가 6개 문자로 되여 있지 않는 경우; 그ID에 대한 예약이 없는 경우;
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CFlightRecord.alreadyExists, checkReservationID, insert
해설	사용자가 입력한 값으로 특정한 예약을 설정한다.

클라스이름	CFlightRecord
방법이름	scanSpecialMeals
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	비행번호가 3자리까지 되여 있지 않는 경우; 날자가 부정확하게 입력되여 있는 경우
호출된 파일	fltRec.dat, tempF.dat
변경된 파일	fltRec.dat, tempF.dat
호출된 모듈	CFlightRecord.checkFlightNum, insert
해설	특정한 비행에 대한 모든 예약에 대하여(비행번호+비행날자) 식사가 올랐는가를 사용자에게 질문하고 파일을 갱신한다.

클라스이름	CFlightRecord
방법이름	seatReserved
귀환값형	**boolean**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	자석번호가 이미 다른 사람에게 예약되여 있으면 **TRUE**를 귀환한다.

클라스이름	CLowSodiumReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	비행번호, 비행날자, 식사질을 출력한다.

클라스이름	CNotLoadedReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서에 지적된 날자범위에 있는 저염식사를 포함했으면 이 보고서에 대한 기록을 변경한다.

클라스이름	CNotLoadedReport
방법이름	alreadyEncountered
귀환값형	**boolean**
입력변수	ID_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	notLoaded.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	승객ID가 이미 이 보고서에 들어 있는가를 결정한다(중복은 피한다.).

클라스이름	CnotLoadedReport
방법이름	markEncountered
귀환값형	**void**
입력변수	ID_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	notLoaded.dat, tempNot.dat
변경된 파일	notLoaded.dat, tempNot.dat
호출된 모듈	없다.
해설	현재의 승객ID를 noteLoaded파일에 추가한다.

클라스이름	CNotLoadedReport
방법이름	notLoadedMoreThanOnce
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	한번이상 식사가 오르지 않은 승객이 있는가를 결정한다.

클라스이름	CNotLoadedReport
방법이름	print
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	notLoaded.dat
호출된 모듈	CReport.print
해설	기초클라스print방법을 호출하기전에 필요한 파일을 초기화한다.

클라스이름	CNotLoadedReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	CNotLoadedReport.markEncountered CPassenger.getpassenger
해설	이 방법은 승객이름과 주소 그리고 식사가 오르지 않은 날자들을 출력한다.

클라스이름	CNotLoadedReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CNotLoadedReport.alreadyEncountered notLoadedMoreThanOnce
해설	보고서에 지적된 날자범위에 있으며 그 날자범위내에 한번이상 오르지 않은 식사들이 있으면 이 보고서에 대한 기록을 변경한다.

클라스이름	COnBoardReport
방법이름	getQualifications
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	날자가 부정확하게 입력되여 있는 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	탑승보고서에서 비행날자와 비행번호를 검색하고 검증한다.

클라스이름	COnBoardReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	이 기록에 대한 승객과 좌석번호, 식사류형, 검사통을 출력한다.

클라스이름	COnBoardReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	사용자가 명기한대로 비행날자와 비행번호가 되여 있으면 이 보고서에 대한 기록을 변경한다.

클라스이름	Cpassenger
방법이름	AlreadyExists
귀환값형	**boolean**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	현재 대상에 대한 승객ID가 파일에 이미 존재하는가를 결정한다; ID가 존재하면 사용자는 그 파일에 보관되여 있는 값이 리용되였는가를 질문한다.

클라스이름	CPassenger
방법이름	getDescription
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CPassenger.alreadyExists
해설	모든 승객정보를 검색한다.

클라스이름	CPassenger
방법이름	getPassenger
귀환값형	**boolean**
입력변수	ID_TYPE
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat
변경된 파일	없다.
호출된 모듈	없다.
해설	파일로부터 searchID와 등가인 passengerID를 리용하여 승객기록을 적재한다; 승객기록이 이미 적재되여 있으면 **TRUE**를 귀환한다.

클라스이름	CPassenger
방법이름	insert
귀환값형	**void**
입력변수	없다
출력변수	없다.
오유통보문	없다.
호출된 파일	passenger.dat, tempP.dat
변경된 파일	passenger.dat, tempP.dat
호출된 모듈	없다.
해설	파일의 적당한 곳에 승객대상을 삽입한다.

클라스이름	CpercentageRecord
방법이름	Print
귀환값형	**Void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CReport.print
해설	기초클라스print방법을 확장한다; 식사를 추적할 수 있는 배렬을 초기화한다. 그리고 기초클라스 print방법을 호출하고 배렬을 인쇄한다.

클라스이름	CPercentageReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	그 어떤 정보도 인쇄하지 않는다. 서로 다른 종류의 퍼센트를 추적할수 있는 배렬들을 갱신한다.

클라스이름	CPercentageReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightReport
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서의 해당한 날자범위내에 특별식사가 있다면 이 보고서에 대한 기록을 변경한다.

클라스이름	CPoorQualityReport
방법이름	printRecord
귀환값형	**void**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	CPassenger.getPassenger
해설	승객, 비행날자, 식사류형을 출력한다.

클라스이름	CPoorQualityReport
방법이름	qualifiesForReport
귀환값형	**boolean**
입력변수	CFlightRecord
출력변수	없다.
오유통보문	없다.
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	보고서에 지적된 날자범위내에 특별식사가 올라 있지 않으면 이 보고서에 대한 기록을 변경한다.

클라스이름	CReport
방법이름	getQualifications
귀환값형	**void**
입력변수	없다.
출력변수	없다.
오유통보문	날자가 부정확하게 입력되여 있는 경우; 마감날자가 시작날자보다 더 앞선 경우
호출된 파일	없다.
변경된 파일	없다.
호출된 모듈	없다.
해설	변경을 위한 암묵적인 방법; 즉 보고서에 지적된 범위에 대한 시작날자와 마감날자를 얻는다.

클라스이름	Creport
방법이름	Print
귀환값형	**Void**
입력변수	없다.
출력변수	없다.
오유통보문	없다.
호출된 파일	fltRec.dat
변경된 파일	없다.
호출된 모듈	CReport.getQualifications, printRecord, QualifiesForReport
해설	보고서를 인쇄하는 암묵적인 방법; 즉 비행기록 파일에 있는 모든 기록을 반복한다; 보고서에 한정되여 있는 매개 기록에 대하여 함수printRecord를 호출한다.

부록 1 0. 항공음식전문회사 실례연구 : 검은통시험실례

항공음식전문회사제품을 위한 검은통시험실례의 대표적인 실례는 경계분석과 기능분석을 리용하여 작성되였다.

경계값분석

날자자료

날자에 대한 등가클라스 :

1.	적당한 위치에서의 /기호의 탈락	오유
2.	날자요소에서 선두령기호의 탈락 (실례로 MAR/6/1998 대 MAR/06/1998)	오유
3.	달이름요소가 타당한 세문자략어가 아닌 경우	오유
4.	날자요소< 1	오유
5.	날자요소>현재 달의 일수 (윤년에 대한 시험을 여기서 진행한다.)	오유
6.	년요소< 0	오유
7.	년요소> 2999	오유
8.	MMM/DD/YYYY형식의 타당한 날자	수용가능

승객자료

passengerID에 대한 등가클라스:

1.	<1개 문자	오유
2.	>9개 문자	오유
3.	9개 문자	수용가능

세례명(firstName), 이름(lastName)에 대한 등가클라스:

1.	<1개 문자	오유
2.	1개 문자	수용가능
3.	1개 문자로부터 15개 문자사이	수용가능
4.	15개 문자	수용가능
5.	>15개 문자	기록하지 않음

성의 첫 글자(middleInit)에 대한 등가클라스:

1.	<1개 문자	수용가능
2.	1개 문자	수용가능
3.	>1개 문자	기록하지 않음

뒤붙이(suffix)에 대한 등가클라스:

1.	<1개 문자	수용가능
2.	1개 문자	수용가능
3.	1개 문자와 5개 문자사이	수용가능
4.	5개 문자	수용가능
5.	>5개 문자	기록하지 않음

주소1(address1)에 대한 등가클라스:

1.	<1개 문자	오유
2.	1개 문자	수용가능
3.	1개 문자와 25개 문자사이	수용가능
4.	25개 문자	수용가능
5.	>25개 문자	기록하지 않음

주소2(address2)에 대한 등가클라스:

1.	<1개 문자	수용가능
2.	1개 문자	수용가능
3.	1개 문자와 25개 문자사이	수용가능
4.	25개 문자	수용가능
5.	>25개 문자	기록하지 않음

도시(city)에 대한 등가클라스:

1.	<1개 문자	오유
2.	1개 문자	수용가능
3.	1개 문자와 14개 문자사이	수용가능
4.	14개 문자	수용가능
5.	>14개 문자	기록하지 않음

나라(state)에 대한 등가클라스:

1.	<1개 문자	수용가능
2.	1개 문자	수용가능
3.	1개 문자와 14개 문자사이	수용가능
4.	14개 문자	수용가능
5.	>14개 문자	기록하지 않음

우편대호(postalCode)에 대한 등가클라스:

1.	>1개 문자	수용가능
2.	1개 문자	수용가능
3.	1개 문자와 10개 문자	수용가능
4.	10개 문자	수용가능
5.	>10개 문자	기록하지 않음

고향(country)에 대한 등가클라스:

1.	>1개 문자	오유
2.	1개 문자	수용가능
3.	1개 문자와 20개 문자	수용가능
4.	20개 문자	수용가능
5.	>20개 문자	기록하지 않음

기능분석

예약작성

1. reservationID가 이미 존재한다는 예약작성
2. 어떤 특별비행을 위한 좌석번호가 이미 예약된다는 예약작성
3. passengerID가 자료기지에 이미 존재한다는 예약작성
4. passengerID가 자료기지에 이미 존재하지 않는다는 예약작성

승객검사

5. preservationID를 자료기지에서 찾을수 없는 승객을 검사
6. reservationID를 자료기지에서 찾을수 있는 승객을 검사

특별식사목록의 조사

7. 자료기지에 존재하지 않는 비행날자와 비행기번호에 대하여 특별식사목록을 조사
8. 자료기지에 존재하는 비행날자와 비행기번호에 대하여 특별식사목록을 조사

우편엽서의 조사

9. reservationID자료기지에서 찾을수 없는 반환된 우편엽서를 조사
10. reservationID를 자료기지에서 찾을수 있는 반환된 우편엽서를 조사

보고서 현시

11. 매 보고서에 대하여 출발날자(실례로 MAR/10/1995)가 마감날자(실례로 MAR/10/1990)보다 더 늦어 지는 보고서를 현시
12. 조달자와 탑승보고서에 대하여 자료기지에 존재하지 않는 비행날자와 비행기번호에 대한 보고서를 현시
13. 25h 조달자 목록의 현시
14. 탑승식사목록의 현시
15. 주문한 식사가 한번이상 오르지 않은 승객이 없을 때 비적재보고서를 현시
16. 주문한 식사가 한번이상 오르지 않은 승객이 여러명일 때 비적재보고서를 현시(같은 날자 여러명의 예약이 발행할 때에도 우의 작용을 시도한다.)
17. 품질이 5급이하라고 생각되는 식사가 오른 경우의 승객이 자료기지안

에 없을 때 저품질식사에 대한 보고서를 현시

18. 품질이 5급이하라고 생각되는 식사가 오른 경우의 승객이 자료기지안에 여러명일 때 저품질식사에 대한 보고서를 현시
19. 퍼센트에 대한 보고서를 현시
20. 자료기지에 저염식사항목이 없을 때 저염식사에 대한 보고서를 현시
21. 자료지안에 저염식사항목이 여러개 있을 때 저염식사에 대한 보고서를 현시

부록 11. 항공음식전문회사 실례연구 : C++원천코드

항공음식전문회사제품을 위한 완전한 C++원천코드는 World Wide Web의 www.mhhe.com/engcs/compsci/schach에서 리용할수 있다.

부록 12. 항공음식전문회사 실례연구 : JAVA원천코드

항공음식전문회사제품을 위한 완전한 Java원천코드는 World Wide Web의 www.mhhe.com/engcs/compsci/schach에서 리용할수 있다.

참 고 문 헌

문헌을 인용한 장이 ( )안에 지적되어 있다.

[Abrial, 1980] J.-R. ABRIAL, "The Specification Language Z: Syntax and Semantics," Oxford University Computing Laboratory, Programming Research Group, Oxford, U.K., April 1980. (Chapter 11)

[Ackerman, Buchwald, and Lewski, 1989] A. F. ACKERMAN, L. S. BUCHWALD, AND F. H. LEWSKI, "Software Inspections: An Effective Verification Process," *IEEE Software* **6** (May 1989), pp. 31–36. (Chapter 6)

[AdaIC, 1997] "DoD to Replace Ada Mandate with Software-Engineering Process," *AdaIC News* (Summer 1997), Ada Information Clearinghouse, Falls Church, VA. (Chapter 8)

[Adler, 1995] R. M. ADLER, "Emerging Standards for Component Software," *IEEE Computer* **28** (March 1995), pp. 68–77. (Chapter 8)

[Adolph, 1996] W. S. ADOLPH, "Cash Cow in the Tar Pit: Reengineering a Legacy System," *IEEE Software* **13** (May 1996), pp. 41–47. (Chapter 16)

[Albrecht, 1979] A. J. ALBRECHT, "Measuring Application Development Productivity," *Proceedings of the IBM SHARE/GUIDE Applications Development Symposium,* Monterey, CA, October 1979, pp. 83–92. (Chapter 9)

[Alexander, 1999] C. ALEXANDER, "The Origins of Pattern Theory," *IEEE Software* **16** (September/October 1999), pp. 71–82. (Chapter 8)

[Alexander et al., 1977] C. ALEXANDER, S. ISHIKAWA, M. SILVERSTEIN, M. JACOBSON, I. FIKSDAHL-KING, AND S. ANGEL, *A Pattern Language,* Oxford University Press, New York, 1977. (Chapter 8)

[Alford, 1985] M. ALFORD, "SREM at the Age of Eight; The Distributed Computing Design System," *IEEE Computer* **18** (April 1985), pp. 36–46. (Chapter 11)

[Anderson et al., 1995] E. ANDERSON, Z. BAI, C. BISCHOF, J. DEMMEL, J. DONGARRA, J. DU CROZ, A. GREENBAUM, S. HAMMARLING, A. MCKENNEY, S. OSTROUCHOV, AND D. SORENSEN, *LAPACK Users' Guide,* 2nd ed., SIAM, Philadelphia, 1995. (Chapter 8)

[ANSI X3.159, 1989] "The Programming Language C," ANSI X3.159-1989, American National Standards Institute, New York, 1989. (Chapter 8)

[ANSI/IEEE 754, 1985] "Standard for Binary Floating Point Arithmetic," ANSI/IEEE 754, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1985. (Chapter 8)

[ANSI/IEEE 770X3.97, 1983] "Pascal Computer Programming Language," ANSI/IEEE 770X3.97-1983, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1983. (Chapter 8)

[ANSI/IEEE 829, 1991] "Software Test Documentation," ANSI/IEEE 829-1991, American National Standards Institute, Institute of Electrical and Electronic Engineers, New York, 1991. (Chapter 9)

[ANSI/MIL-STD-1815A, 1983] "Reference Manual for the Ada Programming Language," ANSI/MIL-STD-1815A, American National Standards Institute, United States Department of Defense, Washington, DC, 1983. (Chapter 8)

[Aoyama, 1993] M. AOYAMA, "Concurrent-Development Process Model," *IEEE Computer* **10** (July 1993), pp. 46–55. (Chapter 3)

[Apple, 1984] *Macintosh Pascal User's Guide,* Apple Computer, Inc., Cupertino, CA, 1984. (Chapter 5)

[Arthur, 1997] L. J. ARTHUR, "Quantum Improvements in Software System Quality," *Communications of the ACM* **40** (June 1997), pp. 46–52. (Chapter 6)

[Awad, Kuusela, and Ziegler, 1996] M. AWAD, J. KUUSELA, AND J. ZIEGLER, *Object-Oriented Technology for Real-Time Systems,* Prentice Hall, Upper Saddle River, NJ, 1996. (Chapter 12)

[Baber, 1987] R. L. BABER, *The Spine of Software: Designing Provably Correct Software: Theory and Practice,* John Wiley and Sons, New York, 1987. (Chapter 6)

[Babich, 1986] W. A. BABICH, *Software Configuration Management: Coordination for Team Productivity,*

Addison-Wesley, Reading, MA, 1986. (Chapter 5)

[Baetjer, 1996] H. BAETJER, *Software as Capital: An Economic Perspective on Software Engineering,* IEEE Computer Society Press, Los Alamitos, CA, 1996. (Chapter 1)

[Baker, 1972] F. T. BAKER, "Chief Programmer Team Management of Production Programming," *IBM Systems Journal* **11** (No. 1, 1972), pp. 56–73. (Chapter 4)

[Balzer, 1985] R. BALZER, "A 15 Year Perspective on Automatic Programming," *IEEE Transactions on Software Engineering* **SE-11** (November 1985), pp. 1257–68. (Chapter 11)

[Bamberger, 1997] J. BAMBERGER, "Essence of the Capability Maturity Model," *IEEE Computer* **30** (June 1997), pp. 112–14. (Chapter 2)

[Bamford and Deibler, 1993a] R. C. BAMFORD AND W. J. DEIBLER, II, "Comparing, Contrasting ISO 9001 and the SEI Capability Maturity Model," *IEEE Computer* **26** (October 1993), pp. 68–70. (Chapter 2)

[Bamford and Deibler, 1993b] R. C. BAMFORD AND W. J. DEIBLER, II, "A Detailed Comparison of the SEI Software Maturity Levels and Technology Stages to the Requirements for ISO 9001 Registration," Software Systems Quality Consulting, San Jose, CA, 1993. (Chapter 2)

[Banker, Datar, Kemerer, and Zwieg, 1993] R. D. BANKER, S. M. DATAR, C. F. KEMERER, AND D. ZWIEG, "Software Complexity and Maintenance Costs," *Communications of the ACM* **36** (November 1993), pp. 81–94. (Chapter 16)

[Banks, Carson, Nelson, and Nichol, 2000] J. BANKS, J. S. CARSON, B. L. NELSON, AND D. M. NICHOL, *Discrete-Event System Simulation,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1995. (Chapter 11)

[Barnard and Price, 1994] J. BARNARD AND A. PRICE, "Managing Code Inspection Information," *IEEE Software* **11** (March 1994), pp. 59–69. (Chapter 14)

[Barnes and Bollinger, 1991] B. H. BARNES AND T. B. BOLLINGER, "Making Reuse Cost-Effective," *IEEE Software* **8** (January 1991), pp. 13–24. (Chapter 8)

[Basili, 1990] V. R. BASILI, "Viewing Maintenance as Reuse-Oriented Software Development," *IEEE Software* **7** (January 1990), pp. 19–25. (Chapter 16)

[Basili and Hutchens, 1983] V. R. BASILI AND D. H. HUTCHENS, "An Empirical Study of a Syntactic Complexity Family," *IEEE Transactions on Software Engineering* **SE-9** (November 1983), pp. 664–72. (Chapter 14)

[Basili and Selby, 1987] V. R. BASILI AND R. W. SELBY, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Transactions on Software Engineering* **SE-13** (December 1987), pp. 1278–96. (Chapter 14)

[Basili and Weiss, 1984] V. R. BASILI AND D. M. WEISS, "A Methodology for Collecting Valid Software Engineering Data," *IEEE Transactions on Software Engineering* **SE-10** (November 1984), pp. 728–38. (Chapter 14)

[Basili et al., 1995] V. BASILI, M. ZELKOWITZ, F. MCGARRY, J. PAGE, S. WALIGORA, AND R. PAJERSKI, "SEL's Software Process-Improvement Program," *IEEE Software* **12** (November 1995), pp. 83–87. (Chapter 2)

[Bass, Clements, and Kazman, 1998] L. BASS, P. CLEMENTS, AND R. KAZMAN, *Software Architecture in Practice,* Addison-Wesley, Reading, MA, 1998. (Chapter 8)

[Beck, 1999] K. BECK, "Embracing Change with Extreme Programming," *IEEE Computer* **32** (October 1999), pp. 70–77. (Chapters 3 and 4)

[Beck, 2000] K. BECK, *Extreme Programming Explained: Embrace Change,* Addison Wesley Longman, Reading, MA, 2000. (Chapters 3 and 4)

[Beck and Cunningham, 1989] K. BECK AND W. CUNNINGHAM, "A Laboratory for Teaching Object-Oriented Thinking," Proceedings of OOPSLA '89, *ACM SIGPLAN Notices* **24** (October 1989), pp. 1–6. (Chapter 12)

[Beizer, 1990] B. BEIZER, *Software Testing Techniques,* 2nd ed., Van Nostrand Reinhold, New York, 1990. (Chapters 2, 6, 13, and 14)

[Beizer, 1995] B. BEIZER, *Black-Box Testing: Techniques for Functional Testing of Software and Systems,* John Wiley and Sons, New York, 1995. (Chapter 14)

[Beizer, 1997] B. BEIZER, "Cleanroom Process Model: A Critical Examination," *IEEE Software* **14** (March/April 1997), pp. 14–16. (Chapter 14)

[Bellinzona, Fugini, and Pernici, 1995] R. BELLINZONA, M. G. FUGINI, AND B. PERNICI, "Reusing Specifications in OO Applications," *IEEE Software* **12** (March 1995), pp. 656–75. (Chapter 12)

[Berard, 1993] E. V. BERARD, *Essays on Object-Oriented Software Engineering,* Volume 1, Prentice Hall, Englewood Cliffs, NJ, 1993. (Chapter 3)

[Berry, 1978] D. M. BERRY, personal communication, 1978. (Chapter 7)

[Berry and Wing, 1985] D. M. BERRY AND J. M. WING, "Specifying and Prototyping: Some Thoughts on Why They Are Successful," in: *Formal Methods and Software Development, Proceedings of the International Joint Conference on Theory and Practice of Software Development,* Volume 2, Springer-Verlag, Berlin, 1985, pp. 117–28. (Chapter 6)

[Bersoff and Davis, 1991] E. H. BERSOFF AND A. M. DAVIS, "Impacts of Life Cycle Models on Software Configuration Management," *Communications of the ACM* **34** (August 1991), pp. 104–18. (Chapter 5)

[Bhandari et al., 1994] I. BHANDARI, M. J. HALLIDAY, J. CHAAR, R. CHILLAREGE, K. JONES, J. S. ATKINSON, C. LEPORI-COSTELLO, P. Y. JASPER, E. D. TARVER, C. C. LEWIS, AND M. YONEZAWA, "In-Process Improvement through Defect Data Interpretation," *IBM Systems Journal* **33** (No. 1, 1994), pp. 182–214. (Chapter 1)

[Bias, 1991] R. BIAS, "Walkthroughs: Efficient Collaborative Testing," *IEEE Software* **8** (September 1991), pp. 94–95. (Chapter 13)

[Binkley and Schach, 1996] A. B. BINKLEY AND S. R. SCHACH, "A Comparison of Sixteen Quality Metrics for Object-Oriented Design," *Information Processing Letters* **57,** (No. 6, June 1996), pp. 271–75. (Chapters 7 and 13)

[Binkley and Schach, 1997] A. B. BINKLEY AND S. R. SCHACH, "Toward a Unified Approach to Object-Oriented Coupling," *Proceedings of the 35th Annual ACM Southeast Conference,* Murfreesboro, TN, April 2–4, 1997, pp. 91–97. (Chapters 7 and 13)

[Binkley and Schach, 1998] A. B. BINKLEY AND S. R. SCHACH, "Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures," *Proceedings of the 20th International Conference on Software Engineering,* Kyoto, Japan, April 1998, pp. 542–55. (Chapter 13)

[Bisbal, Lawless, Wu, and Grimson, 1999] J. BISBAL, D. LAWLESS, B. WU, J. GRIMSON, "Legacy Information Systems: Issues and Directions," *IEEE Software* **16** (September/October 1999), pp. 103–111. (Chapter 16)

[Blaha, Premerlani, and Rumbaugh, 1988] M. R. BLAHA, W. J. PREMERLANI, AND J. E. RUMBAUGH, "Relational Database Design Using an Object-Oriented Methodology," *Communications of the ACM* **31** (April 1988), pp. 414–27. (Chapter 7)

[Boehm, 1976] B. W. BOEHM, "Software Engineering," *IEEE Transactions on Computers* **C-25** (December 1976), pp. 1226–41. (Chapters 1 and 2)

[Boehm, 1979] B. W. BOEHM, "Software Engineering, R & D Trends and Defense Needs," in *Research Directions in Software Technology,* P. Wegner (Editor), The MIT Press, Cambridge, MA, 1979. (Chapter 1)

[Boehm, 1980] B. W. BOEHM, "Developing Small-Scale Application Software Products: Some Experimental Results," *Proceedings of the Eighth IFIP World Computer Congress,* October 1980, pp. 321–26. (Chapter 1)

[Boehm, 1981] B. W. BOEHM, *Software Engineering Economics,* Prentice Hall, Englewood Cliffs, NJ, 1981. (Chapters 1 and 9)

[Boehm, 1984a] B. W. BOEHM, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software* **1** (January 1984), pp. 75–88. (Chapter 6)

[Boehm, 1984b] B. W. BOEHM, "Software Engineering Economics," *IEEE Transactions on Software Engineering* **SE-10** (January 1984), pp. 4–21. (Chapter 9)

[Boehm, 1988] B. W. BOEHM, "A Spiral Model of Software Development and Enhancement," *IEEE Computer* **21** (May 1988), pp. 61–72. (Chapter 3)

[Boehm, 1991] B. W. BOEHM, "Software Risk Management: Principles and Practices," *IEEE Software* **8** (January 1991), pp. 32–41. (Chapter 3)

[Boehm, 1997] R. BOEHM (EDITOR), "Function Point FAQ," ourworld.compuserve.com/homepages/softcomp/fpfaq.htm, June 25, 1997. (Chapter 9)

[Boehm et al., 1984] B. W. BOEHM, M. H. PENEDO, E. D. STUCKLE, R. D. WILLIAMS, AND A. B. PYSTER, "A Software Development Environment for Improving Productivity," *IEEE Computer* **17** (June 1984), pp. 30–44. (Chapter 3 and 9)

[Boehm et al., 2000] B. W. BOEHM, C. ABTS, A. W. BROWN, S. CHULANI, B. K. CLARK, E. HOROWITZ, R. MADACHY, D. REIFER, AND B. STEECE, *Software Cost Estimation with COCOMO II,* Prentice Hall, Upper Saddle River, NJ, 2000. (Chapter 9)

[Boehm, Gray, and Seewaldt, 1984] B. W. BOEHM, T. E. GRAY, AND T. SEEWALDT, "Prototyping versus

Specifying: A Multi-Project Experiment," *IEEE Transactions on Software Engineering* **SE-10** (May 1984), pp. 290–303. (Chapter 10)

[Bohrer, Johnson, Nilsson, and Rubin, 1998] K. BOHRER, V. JOHNSON, A. NILSSON, AND B. RUBIN, "Business Process Components for Distributed Object Applications," *Communications of the ACM* **41** (June 1998), pp. 43–48. (Chapter 8)

[Bollinger and McGowan, 1991] T. BOLLINGER AND C. MCGOWAN, "A Critical Look at Software Capability Evaluations," *IEEE Software* **8** (July 1991), pp. 25–41. (Chapter 2)

[Boloix and Robillard, 1995] G. BOLOIX AND P. N. ROBILLARD, "A Software System Evaluation Framework," *IEEE Computer* **28** (December 1995), pp. 17–26. (Chapter 6)

[Booch, 1994] G. BOOCH, *Object-Oriented Analysis and Design with Applications,* 2nd ed., Benjamin/Cummings, Redwood City, CA, 1994. (Chapters 3 and 12)

[Booch, Rumbaugh, and Jacobson, 1999] G. BOOCH, J. RUMBAUGH, AND I. JACOBSON, *The UML Users Guide,* Addison-Wesley, Reading, MA, 1999. (Chapter 12)

[Bosch, 2000] J. BOSCH, *Design and Use of Software Architectures,* Addison-Wesley, Reading, MA, 2000. (Chapter 8)

[Bowen and Hinchey, 1995a] J. P. BOWEN AND M. G. HINCHEY, "Ten Commandments of Formal Methods," *IEEE Computer* **28** (April 1995), pp. 56–63. (Chapter 11)

[Bowen and Hinchey, 1995b] J. P. BOWEN AND M. G. HINCHEY, "Seven More Myths of Formal Methods," *IEEE Software* **12** (July 1995), pp. 34–41. (Chapter 11)

[Box, 1998] D. BOX, *Essential COM,* Addison-Wesley, Reading, MA, 1998. (Chapter 8)

[Brady, 1977] J. M. BRADY, *The Theory of Computer Science,* Chapman and Hall, London, 1977. (Chapter 11)

[Brandl, 1990] D. L. BRANDL, "Quality Measures in Design: Finding Problems before Coding," *ACM SIGSOFT Software Engineering Notes* **15** (January 1990), pp. 68–72. (Chapter 13)

[Brereton et al., 1999] P. BRERETON, D. BUDGEN, K. BENNETT, M. MUNRO, P. LAYZELL, L. MACAULAY, D. GRIFFITHS, AND C. STANNETT, "The Future of Software," *Communications of the ACM* **42** (December 1999), pp. 78–84. (Chapter 1)

[Brettschneider, 1989] R. BRETTSCHNEIDER, "Is Your Software Ready for Release?" *IEEE Software* **6** (July 1989), pp. 100, 102, and 108. (Chapter 15)

[Brodman and Johnson, 1996] J. G. BRODMAN AND D. JOHNSON, "Return on Investment from Software Process Improvement as Measured by U.S. Industry," *CrossTalk* **9** (April 1996), pp. 23–28. (Chapter 2)

[Brooks, 1975] F. P. BROOKS, JR., *The Mythical Man-Month: Essays on Software Engineering,* Addison-Wesley, Reading, MA, 1975. Twentieth Anniversary Edition, Addison-Wesley, Reading, MA, 1995. (Chapters 1, 2, 4, and 10)

[Brooks, 1986] F. P. BROOKS, JR., "No Silver Bullet," in: *Information Processing '86,* H.-J. Kugler (Editor), Elsevier North-Holland, New York, 1986. Reprinted in *IEEE Computer* **20** (April 1987), pp. 10–19. (Chapters 2 and 13)

[Brown, 1996] A. W. BROWN, *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute,* IEEE Computer Society Press, Los Alamitos, CA, 1996. (Chapter 8)

[Brown, 1999] A. W. BROWN, "Mastering the Middleware Muddle," *IEEE Software* **16** (July/August 1999), pp. 18–21. (Chapter 8)

[Brown and McDermid, 1992] A. W. BROWN AND J. A. MCDERMID, "Learning from IPSE's Mistakes," *IEEE Software* **9** (March 1992), pp. 23–29. (Chapter 15)

[Brown et al., 1998] W. J. BROWN, R. C. MALVEAU, W. H. BROWN, H. W. MCCORMICK, III, AND T. J. MOWBRAY, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis,* John Wiley and Sons, New York, 1998. (Chapter 8)

[Browne, 1994] D. BROWNE, *STUDIO: STructured User-interface Design for Interaction Optimization,* Prentice Hall, Englewood Cliffs, NJ, 1994. (Chapter 10)

[Bruegge, Blythe, Jackson, and Shufelt, 1992] B. BRUEGGE, J. BLYTHE, J. JACKSON, AND J. SHUFELT, "Object-Oriented Modeling with OMT," *Proceedings of the Conference on Object-Oriented Programming, Languages, and Systems, OOPSLA '92, ACM SIGPLAN Notices* **27** (October 1992), pp. 359–376. (Chapter 7)

[Bruno and Marchetto, 1986] G. BRUNO AND G. MARCHETTO, "Process-Translatable Petri Nets for the Rapid Prototyping of Process Control Systems," *IEEE Transactions on Software*

Engineering **SE-12** (February 1986), pp. 346–57. (Chapter 11)

[Budd, 1991] T. A. BUDD, *An Introduction to Object-Oriented Programming,* Addison-Wesley, Reading, MA, 1991. (Chapter 1)

[Bush, 1990] M. BUSH, "Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory," *Proceedings of the 12th International Conference on Software Engineering,* Nice, France, March 1990, pp. 196–99. (Chapter 6)

[Business Week Online, 1999] Business Week Online, February 2, 1999, www.businessweek.com/1999/99_08/b3617025.htm. (Chapter 4)

[Caldiera and Basili, 1991] G. CALDIERA AND V. R. BASILI, "Identifying and Qualifying Reusable Software Components," *IEEE Computer* **24** (February 1991), pp. 61–70. (Chapter 8)

[Capper, Colgate, Hunter, and James, 1994] N. P. CAPPER, R. J. COLGATE, J. C. HUNTER, AND M. F. JAMES, "The Impact of Object-Oriented Technology on Software Quality: Three Case Histories," *IBM Systems Journal* **33** (No. 1, 1994), pp. 131–57. (Chapters 1, 7, and 10)

[Card, 2000] D. N. CARD, 'Sorting Out Six Sigma and the CMM," *IEEE Software* **14** (May/June 2000), pp. 11–13. (Chapter 2)

[Carlson, Druffel, Fisher, and Whitaker, 1980] W. E. CARLSON, L. E. DRUFFEL, D. A. FISHER, AND W. A. WHITAKER, "Introducing Ada," *Proceedings of the ACM Annual Conference, ACM 80,* Nashville, TN, 1980, pp. 263–71. (Chapter 8)

[Charette, 1996] R. N. CHARETTE, "Large-Scale Project Management *Is* Risk Management," *IEEE Software* **13** (July 1996), pp. 110–17. (Chapter 9)

[Charette, Adams, and White, 1997] R. N. CHARETTE, K. M. ADAMS, AND M. B. WHITE, "Managing Risk in Software Maintenance," *IEEE Software* **14** (May/June 1997), pp. 43–50. (Chapter 16)

[Chen, 1976] P. CHEN, "The Entity-Relationship Model—Towards a Unified View of Data," *ACM Transactions on Database Systems* **1** (March 1976), pp. 9–36. (Chapter 11)

[Chen and Norman, 1992] M. CHEN AND R. J. NORMAN, "A Framework for Integrated CASE," *IEEE Software* **8** (March 1992), pp. 18–22. (Chapter 15)

[Chidamber and Kemerer, 1994] S. R. CHIDAMBER AND C. F. KEMERER, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering* **20** (June 1994), pp. 476–93. (Chapters 7 and 13)

[Chmura and Crockett, 1995] A. CHMURA AND H. D. CROCKETT, "What's the Proper Role for CASE Tools?" *IEEE Software* **12** (March 1995), pp. 18–20. (Chapter 5)

[Clarke, Podgurski, Richardson, and Zeil, 1989] L. A. CLARKE, A. PODGURSKI, D. J. RICHARDSON, AND S. J. ZEIL, "A Formal Evaluation of Data Flow Path Selection Criteria," *IEEE Transactions on Software Engineering* **15** (November 1989), pp. 1318–32. (Chapter 14)

[Cline, 1996] M. P. CLINE, "The Pros and Cons of Adopting and Applying Design Patterns in the Real World," *Communications of the ACM* **39** (October 1996), pp. 47–49. (Chapter 8)

[Coad, 1992] P. COAD, "Object-Oriented Patterns," *Communications of the ACM* **35** (September 1992), pp. 152–59. (Chapter 8)

[Coad and Yourdon, 1991a] P. COAD AND E. YOURDON, *Object-Oriented Analysis,* 2nd ed., Yourdon Press, Englewood Cliffs, NJ, 1991. (Chapters 3 and 12)

[Coad and Yourdon, 1991b] P. COAD AND E. YOURDON, *Object-Oriented Design,* Yourdon Press, Englewood Cliffs, NJ, 1991. (Chapter 13)

[Coleman, Ash, Lowther, and Oman, 1994] D. COLEMAN, D. ASH, B. LOWTHER, AND P. OMAN, "Using Metrics to Evaluate Software System Maintainability," *IEEE Computer* **27** (August 1994), pp. 44–49. (Chapter 1)

[Coleman, Hayes, and Bear, 1992] D. COLEMAN, F. HAYES, AND S. BEAR, "Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design," *IEEE Transactions on Software Engineering* **18** (January 1992), pp. 9–18. (Chapters 11 and 12)

[Coleman et al., 1994] D. COLEMAN, P. ARNOLD, S. BODOFF, C. DOLLIN, H. GILCHRIST, F. HAYES, AND P. JEREMAES, *Object-Oriented Development: The Fusion Method,* Prentice Hall, Englewood Cliffs, NJ, 1994. (Chapter 12)

[Connell and Shafer, 1989] J. L. CONNELL AND L. SHAFER, *Structured Rapid Prototyping: An Evolutionary Approach to Software Development,* Yourdon Press, Englewood Cliffs, NJ, 1989. (Chapters 3 and 10)

[Coolahan and Roussopoulos, 1983] J. E. COOLAHAN, JR., AND N. ROUSSOPOULOS, "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets," *IEEE Transactions on Software*

Engineering **SE-9** (September 1983), pp. 603–16. (Chapter 11)

[Cooling, 1997] J. E. COOLING, *Real-Time Software Systems: An Introduction,* Van Nostrand Reinhold, New York, 1997. (Chapter 13)

[Cooper, 1998] J. W. COOPER, "Using Design Patterns," *Communications of the ACM* **42** (June 1998), pp. 65–68. (Chapter 8)

[Coplien, 1997] J. O. COPLIEN, "Idioms and Patterns as Architectural Literature," *IEEE Software* **14** (January/February 1997), pp. 36–42. (Chapter 8)

[Coulter, 1983] N. S. COULTER, "Software Science and Cognitive Psychology," *IEEE Transactions on Software Engineering* **SE-9** (March 1983), pp. 166–71. (Chapter 5)

[Cox, 1990] B. J. COX, "There *Is* a Silver Bullet," *Byte* **15** (October 1990), pp. 209–18. (Chapter 2)

[Crossman, 1982] T. D. CROSSMAN, "Inspection Teams, Are They Worth It?" *Proceedings of the Second National Symposium on EDP Quality Assurance,* Chicago, November 1982. (Chapter 14)

[Cusumano and Selby, 1995] M. A. CUSUMANO AND R. W. SELBY, *Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets, and Manages People,* The Free Press/Simon and Schuster, New York, 1995. (Chapters 3 and 4)

[Cusumano and Selby, 1997] M. A. CUSUMANO AND R. W. SELBY, "How Microsoft Builds Software," *Communications of the ACM* **40** (June 1997), pp. 53–61. (Chapters 3 and 4)

[Dahl and Nygaard, 1966] O.-J. DAHL AND K. NYGAARD, "SIMULA—An ALGOL-Based Simulation Language," *Communications of the ACM* **9** (September 1966), pp. 671–78. (Chapter 7)

[Daly, 1977] E. B. DALY, "Management of Software Development," *IEEE Transactions on Software Engineering* **SE-3** (May 1977), pp. 229–42. (Chapter 1)

[Dart, Ellison, Feiler, and Habermann, 1987] S. A. DART, R. J. ELLISON, P. H. FEILER, AND A. N. HABERMANN, "Software Development Environments," *IEEE Computer* **20** (November 1987), pp. 18–28. (Chapter 11)

[Date, 1999] C. J. DATE, *An Introduction to Database Systems,* 7th ed, Addison-Wesley, Reading, MA, 1999. (Chapter 14)

[Davis, 1993] A. M. DAVIS, *Software Requirements: Objects, Functions, and States,* rev. ed., Prentice Hall, Englewood Cliffs, NJ, 1993. (Chapter 10)

[Dawood, 1994] M. DAWOOD, "It's Time for ISO 9000," *CrossTalk* (March 1994) pp. 26–28. (Chapter 2)

[de Champeaux and Faure, 1992] D. DE CHAMPEAUX AND P. FAURE, "A Comparative Study of Object-Oriented Analysis Methods," *Journal of Object-Oriented Programming* **5** (March/April 1992), pp. 21–33. (Chapter 12)

[Delisle and Garlan, 1990] N. DELISLE AND D. GARLAN, "A Formal Description of an Oscilloscope," *IEEE Software* **7** (September 1990), pp. 29–36. (Chapter 11)

[Delisle and Schwartz, 1987] N. DELISLE AND M. SCHWARTZ, "A Programming Environment for CSP," Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, *ACM SIGPLAN Notices* **22** (January 1987), pp. 34–41. (Chapter 11)

[DeMarco, 1978] T. DEMARCO, *Structured Analysis and System Specification,* Yourdon Press, New York, 1978. (Chapter 11)

[DeMarco and Lister, 1987] T. DEMARCO AND T. LISTER, *Peopleware: Productive Projects and Teams,* Dorset House, New York, 1987. (Chapters 1 and 4)

[DeMarco and Lister, 1989] T. DEMARCO AND T. LISTER, "Software Development: The State of the Art vs. State of the Practice," *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, May 1989, pp. 271–75. (Chapters 1 and 2)

[DeMillo, Lipton, and Perlis, 1979] R. A. DEMILLO, R. J. LIPTON, AND A. J. PERLIS, "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM* **22** (May 1979), pp. 271–80. (Chapter 6)

[DeMillo, Lipton, and Sayward, 1978] R. A. DEMILLO, R. J. LIPTON, AND F. G. SAYWARD, "Hints on Test Data Selection: Help for the Practicing Programmer," *IEEE Computer* **11** (April 1978), pp. 34–43. (Chapter 6)

[Deming, 1986] W. E. DEMING, *Out of the Crisis,* MIT Center for Advanced Engineering Study, Cambridge, MA, 1986. (Chapter 2)

[DeRemer and Kron, 1976] F. DEREMER AND H. H. KRON, "Programming-in-the-Large versus Programming-in-the-Small," *IEEE Transactions on*

Software Engineering **SE-2** (June 1976), pp. 80–86. (Chapter 5)

[Devenny, 1976] T. DEVENNY, "An Exploratory Study of Software Cost Estimating at the Electronic Systems Division," Thesis No. GSM/SM/765–4, Air Force Institute of Technology, Dayton, OH, 1976. (Chapter 9)

[Diaz and Sligo, 1997] M. DIAZ AND J. SLIGO, "How Software Process Improvement Helped Motorola," *IEEE Software* **14** (September/October 1997), pp. 75–81. (Chapter 2)

[Dijkstra, 1968] E. W. DIJKSTRA, "A Constructive Approach to the Problem of Program Correctness," *BIT* **8** (No. 3, 1968), pp. 174–86. (Chapter 6)

[Dijkstra, 1972] E. W. DIJKSTRA, "The Humble Programmer," *Communications of the ACM* **15** (October 1972), pp. 859–66. (Chapter 6)

[Dijkstra, 1976] E. W. DIJKSTRA, *A Discipline of Programming,* Prentice Hall, Englewood Cliffs, NJ, 1976. (Chapter 5)

[Diller, 1994] A. DILLER, *Z: An Introduction to Formal Methods,* 2nd ed., John Wiley and Sons, Chichester, U.K., 1994. (Chapter 11)

[Dion, 1993] R. DION, "Process Improvement and the Corporate Balance Sheet," *IEEE Software* **10** (July 1993), pp. 28–35. (Chapter 2)

[Dix, Finlay, Abowd, and Beale, 1993] A. DIX, J. FINLAY, G. ABOWD, AND R. BEALE, *Human-Computer Interaction,* Prentice Hall, Englewood Cliffs, NJ, 1993. (Chapter 10)

[DoD, 1987] "Report of the Defense Science Board Task Force on Military Software," Office of the Under Secretary of Defense for Acquisition, Washington, DC, September 1987. (Chapter 2)

[Dongarra, Pozo, and Walker, 1993] J. DONGARRA, R. POZO, AND D. WALKER, "LAPACK++: A Design Overview of Object-Oriented Extensions for High Performance Linear Algebra," *Proceedings of Supercomputing '93,* IEEE Press, New York, 1993, pp. 162–71. (Chapter 8)

[Donohoe, 2000] P. DONOHOE (EDITOR), *Software Product Lines: Experience and Research Directions,* Kluwer Academic Publishers, Boston, 2000. (Chapter 8)

[Doolan, 1992] E. P. DOOLAN, "Experience with Fagan's Inspection Method," *Software—Practice and Experience* **22** (February 1992), pp. 173–82. (Chapters 6 and 11)

[Dooley and Schach, 1985] J. W. M. DOOLEY AND S. R. SCHACH, "FLOW: A Software Development Environment Using Diagrams," *Journal of Systems and Software* **5** (August 1985), pp. 203–19. (Chapter 5)

[D'Souza and Wills, 1999] D. D'SOUZA AND A. WILLS, *Objects, Components, and Frameworks with UML: The Catalysis Approach,* Addison-Wesley, Reading, MA, 1999. (Chapters 8 and 12)

[Dunn, 1984] R. H. DUNN, *Software Defect Removal,* McGraw-Hill, New York, 1984. (Chapter 14)

[Ebert, Matsubara, Pezzé, and Bertelsen, 1997] C. EBERT, T. MATSUBARA, M. PEZZÉ, AND O. W. BERTELSEN, "The Road to Maturity: Navigating between Craft and Science," *IEEE Software* **14** (November/December 1997), pp. 77–88. (Chapters 1 and 2)

[El-Rewini et al., 1995] H. EL-REWINI, S. HAMILTON, Y.-P. SHAN, R. EARLE, S. MCGAUGHEY, A. HELAL, R. BADRACHALAM, A. CHIEN, A. GRIMSHAW, B. LEE, A. WADE, D. MORSE, A. ELMAGRAMID, E. PITOURA, R. BINDER, AND P. WEGNER, "Object Technology," *IEEE Computer* **28** (October 1995), pp. 58–72. (Chapters 1 and 7)

[Elshoff, 1976] J. L. ELSHOFF, "An Analysis of Some Commercial PL/I Programs," *IEEE Transactions on Software Engineering* **SE-2** (June 1976), 113–20. (Chapter 1)

[Embley, Jackson, and Woodfield, 1995] D. W. EMBLEY, R. B. JACKSON, AND S. N. WOODFIELD, "OO Systems Analysis: Is It or Isn't It?" *IEEE Software* **12** (July 1995), pp. 18–33. (Chapter 12)

[Endres, 1975] A. ENDRES, "An Analysis of Errors and Their Causes in System Programs," *IEEE Transactions on Software Engineering* **SE-1** (June 1975), pp. 140–49. (Chapter 14)

[Fagan, 1974] M. E. FAGAN, "Design and Code Inspections and Process Control in the Development of Programs," Technical Report IBM-SSD TR 21.572, IBM Corporation, December 1974. (Chapter 1)

[Fagan, 1976] M. E. FAGAN, "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal* **15** (No. 3, 1976), pp. 182–211. (Chapters 6, 11, and 13)

[Fagan, 1986] M. E. FAGAN, "Advances in Software Inspections," *IEEE Transactions on Software Engineering* **SE-12** (July 1986), pp. 744–51. (Chapters 6 and 13)

[Fayad and Johnson, 1999] M. FAYAD AND R. JOHNSON, *Domain-Specific Application Frameworks: Frameworks Experience by Industry,* John Wiley and Sons, New York, 1999. (Chapter 8)

[Fayad and Laitinen, 1997] M. E. FAYAD AND M. LAITINEN, "Process Assessment Considered Wasteful," *Communications of the ACM* **40** (November 1997), pp. 125–28. (Chapter 2)

[Fayad and Schmidt, 1999] M. FAYAD AND D. C. SCHMIDT, *Building Application Frameworks: Object-Oriented Foundations of Framework Design,* John Wiley and Sons, New York, 1999. (Chapter 8)

[Fayad, Schmidt, and Johnson, 1999] M. FAYAD, D. C. SCHMIDT AND R. JOHNSON, *Implementing Application Frameworks: Object-Oriented Frameworks at Work,* John Wiley and Sons, New York, 1999. (Chapter 8)

[Fayad, Tsai, and Fulghum, 1996] M. E. FAYAD, W.-T. TSAI, AND M. L. FULGHUM, "Transition to Object-Oriented Software Development," *Communications of the ACM* **39** (February 1996), pp. 108–21. (Chapter 7)

[Feldman, 1979] S. I. FELDMAN, "Make—A Program for Maintaining Computer Programs," *Software—Practice and Experience* **9** (April 1979), pp. 225–65. (Chapter 5)

[Fenton and Pfleeger, 1997] N. E. FENTON AND S. L. PFLEEGER, *Software Metrics: A Rigorous and Practical Approach,* 2nd ed., IEEE Computer Society, Los Alamitos, CA, 1997. (Chapter 5)

[Ferguson and Sheard, 1998] J. FERGUSON AND S. SHEARD, "Leveraging Your CMM Efforts for IEEE/EIA 12207," *IEEE Software* **15** (September/October 1998), pp. 23–28. (Chapter 2)

[Ferguson et al., 1997] P. FERGUSON, W. S. HUMPHREY, S. KHAJENOORI, S. MACKE, AND A. MATVYA, "Results of Applying the Personal Software Process," *IEEE Computer* **30** (May 1997), pp. 24–31. (Chapter 2)

[Fichman and Kemerer, 1992] R. G. FICHMAN AND C. F. KEMERER, "Object-Oriented and Conventional Analysis and Design Methodologies: Comparison and Critique," *IEEE Computer* **25** (October 1992), pp. 22–39. (Chapters 12 and 13)

[Fichman and Kemerer, 1997] R. G. FICHMAN AND C. F. KEMERER, "Object Technology and Reuse: Lessons from Early Adopters," *IEEE Computer* **30** (July 1997), pp. 47–57. (Chapters 1 and 8)

[Finkelstein, 2000] A. FINKELSTEIN (EDITOR), *The Future of Software Engineering,* IEEE Computer Society Press, Los Alamitos, CA, 2000. (Chapter 1)

[Finney, 1996] K. FINNEY, "Mathematical Notation in Formal Specification: Too Difficult for the Masses?" *IEEE Transactions on Software Engineering* **22** (February 1996), pp. 158–59. (Chapter 11)

[Fisher, 1976] D. A. FISHER, "A Common Programming Language for the Department of Defense—Background and Technical Requirements," Report P-1191, Institute for Defense Analyses, Alexandria, VA, 1976. (Chapter 8)

[Fitzgerald and O'Kane, 1999] B. FITZGERALD AND T. O'KANE, "A Longitudinal Study of Software Process Improvement," *IEEE Software* **16** (May/June 1999), pp. 37–45. (Chapter 2)

[Flanagan and Loukides, 1997] D. FLANAGAN AND M. LOUKIDES, *Java in a Nutshell: A Desktop Quick Reference,* 2nd ed., O'Reilly and Associates, Sebastopol, CA, 1997. (Chapters 7, 8, and 13)

[Fowler, 1986] P. J. FOWLER, "In-Process Inspections of Workproducts at AT&T," AT&T *Technical Journal* **65** (March/April 1986), pp. 102–12. (Chapter 6)

[Fowler, 1997a] M. FOWLER, *Analysis Patterns: Reusable Object Models,* Addison-Wesley, Reading, MA, 1997. (Chapter 8)

[Fowler, 1997b] M. FOWLER WITH K. SCOTT, *UML Distilled,* Addison-Wesley, Reading, MA, 1997. (Chapter 12)

[Fowler et al., 1999] M. FOWLER WITH K. BECK, J. BRANT, W. OPDYKE, AND D. ROBERTS, *Refactoring: Improving the Design of Existing Code,* Addison-Wesley, Reading, MA, 1999. (Chapter 3)

[Frakes and Fox, 1995] W. B. FRAKES AND C. J. FOX, "Sixteen Questions about Software Reuse," *Communications of the ACM* **38** (June 1995), pp. 75–87. (Chapter 8)

[Fraser and Vaishnavi, 1997] M. D. FRASER AND V. K. VAISHNAVI, "A Formal Specifications Maturity Model," *Communications of the ACM* **40** (May 1999), pp. 95–103. (Chapter 11)

[Fuggetta, 1993] A. FUGGETTA, "A Classification of CASE Technology," *IEEE Computer* **26** (December 1993), pp. 25–38. (Chapter 5)

[Fuggetta and Picco, 1994] A. FUGGETTA AND G. P. PICCO, "An Annotated Bibliography on Software Process Improvement," *ACM SIGSOFT Software Engineering Notes* **19** (July 1995), pp. 66–68. (Chapter 2)

[Furey and Kitchenham, 1997] S. FUREY AND B. KITCHENHAM, "Function Points," *IEEE Software* **14** (March/April 1997), pp. 28–32. (Chapter 9)

[Gamma, Helm, Johnson, and Vlissides, 1995] E. GAMMA, R. HELM, R. JOHNSON, AND J. VLISSIDES, *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, MA, 1995. (Chapter 8)

[Gane, 1989] C. GANE, *Rapid System Development: Using Structured Techniques and Relational Technology,* Prentice Hall, Englewood Cliffs, NJ, 1989. (Chapters 3 and 10)

[Gane and Sarsen, 1979] C. GANE AND T. SARSEN, *Structured Systems Analysis: Tools and Techniques,* Prentice Hall, Englewood Cliffs, NJ, 1979. (Chapters 11 and 13)

[Garlan, Allen, and Ockerbloom, 1995] D. GARLAN, R. ALLEN, AND J. OCKERBLOOM, "Architectural Mismatch: Why Reuse Is So Hard," *IEEE Software* **12** (November 1995), pp. 17–26. (Chapter 8)

[Garman, 1981] J. R. GARMAN, "The 'Bug' Heard 'Round the World," *ACM SIGSOFT Software Engineering Notes* **6** (October 1981), pp. 3–10. (Chapter 6)

[Gause and Weinberg, 1990] D. GAUSE AND G. WEINBERG, *Are Your Lights On? How to Figure out What the Problem Really Is,* Dorset House, New York, 1990. (Chapter 10)

[Gelperin and Hetzel, 1988] D. GELPERIN AND B. HETZEL, "The Growth of Software Testing," *Communications of the ACM* **31** (June 1988), pp. 687–95. (Chapter 6)

[Gemmer, 1997] A. GEMMER, "Risk Management: Moving beyond Process," *IEEE Computer* **30** (May 1997), pp. 33–43. (Chapter 9)

[Gentner and Grudin, 1996] D. R. GENTNER AND J. GRUDIN, "Design Models for Computer-Human Interfaces," *IEEE Computer* **29** (June 1996), pp. 28–35. (Chapter 10)

[Gerald and Wheatley, 1999] C. F. GERALD AND P. O. WHEATLEY, *Applied Numerical Analysis,* 6th ed., Addison-Wesley, Reading, MA, 1999. (Chapter 7)

[Ghezzi and Mandrioli, 1987] C. GHEZZI AND D. MANDRIOLI, "On Eclecticism in Specifications: A Case Study Centered around Petri Nets," *Proceedings of the Fourth International Workshop on Software Specification and Design,* Monterey, CA, 1987, pp. 216–24. (Chapter 11)

[Gibbs, 1994] W. W. GIBBS, "Software's Chronic Crisis," *Scientific American* **271** (September 1994), pp. 86–95. (Chapter 1)

[Gifford and Spector, 1987] D. GIFFORD AND A. SPECTOR, "Case Study: IBM's System/360-370 Architecture," *Communications of the ACM* **30** (April 1987), pp. 292–307. (Chapter 8)

[Gilb, 1988] T. GILB, *Principles of Software Engineering Management,* Addison-Wesley, Wokingham, U.K., 1988. (Chapter 3)

[Glass, 1998] R. L. GLASS, "Is There Really a Software Crisis?" *IEEE Software* **15** (January/February 1998), pp. 104–5. (Chapter 1)

[Goldberg and Robson, 1989] A. GOLDBERG AND D. ROBSON, *Smalltalk-80: The Language,* Addison-Wesley, Reading, MA, 1989. (Chapters 7 and 13)

[Goodenough, 1979] J. B. GOODENOUGH, "A Survey of Program Testing Issues," in: *Research Directions in Software Technology,* P. Wegner (Editor), The MIT Press, Cambridge, MA, 1979, pp. 316–40. (Chapter 6)

[Goodenough and Gerhart, 1975] J. B. GOODENOUGH AND S. L. GERHART, "Toward a Theory of Test Data Selection," *Proceedings of the Third International Conference on Reliable Software,* Los Angeles, 1975, pp. 493–510. Also published in *IEEE Transactions on Software Engineering* **SE-1** (June 1975), pp. 156–73. Revised version: J. B. Goodenough, and S. L. Gerhart, "Toward a Theory of Test Data Selection: Data Selection Criteria," in *Current Trends in Programming Methodology,* Volume 2, R. T. Yeh (Editor), Prentice Hall, Englewood Cliffs, NJ, 1977, pp. 44–79. (Chapters 6 and 11)

[Gordon, 1979] M. J. C. GORDON, *The Denotational Description of Programming Languages: An Introduction,* Springer-Verlag, New York, 1979. (Chapter 11)

[Gordon and Bieman, 1995] V. S. GORDON AND J. M. BIEMAN, "Rapid Prototyping Lessons Learned," *IEEE Software* **12** (January 1995), pp. 85–95. (Chapter 10)

[Grady, 1992] R. B. GRADY, *Practical Software Metrics for Project Management and Process Improvement,* Prentice Hall, Englewood Cliffs, NJ, 1992. (Chapters 5 and 14)

[Grady, 1994] R. B. GRADY, "Successfully Applying Software Metrics," *IEEE Computer* **27** (September 1994), pp. 18–25. (Chapter 1)

[Gramlich, 1997] E. M. GRAMLICH, *A Guide to Benefit–Cost Analysis,* 2nd ed., Waveland Books, Prospect Heights, IL, 1997. (Chapter 5)

[Green, 2000] P. GREEN, "FW: Here's an Update to the Simulated Kangaroo Story," *The Risks Digest* **20** (No. 76, January 23, 2000), catless.ncl.ac.uk/Risks/20.76.html. (Chapter 8)

[Griss, 1993] M. L. GRISS, "Software Reuse: From Library to Factory," *IBM Systems Journal* **32** (No. 4, 1993), pp. 548–66. (Chapter 8)

[Guha, Lang, and Bassiouni, 1987] R. K. GUHA, S. D. LANG, AND M. BASSIOUNI, "Software Specification and Design Using Petri Nets," *Proceedings of the Fourth International Workshop on Software Specification and Design,* Monterey, CA, April 1987, pp. 225–30. (Chapter 11)

[Guimaraes, 1985] T. GUIMARAES, "A Study of Application Program Development Techniques," *Communications of the ACM* **28** (May 1985), pp. 494–99. (Chapter 14)

[Guinan, Cooprider, and Sawyer, 1997] P. J. GUINAN, J. G. COOPRIDER, AND S. SAWYER, "The Effective Use of Automated Application Development Tools," *IBM Systems Journal* **36** (No. 1, 1997), pp. 124–39. (Chapter 5)

[Guttag, 1977] J. GUTTAG, "Abstract Data Types and the Development of Data Structures," *Communications of the ACM* **20** (June 1977), pp. 396–404. (Chapter 7)

[Haley, 1996] T. J. HALEY, "Raytheon's Experience in Software Process Improvement," *IEEE Software* **13** (November 1996), pp. 33–41. (Chapter 2)

[Hall, 1990] A. HALL, "Seven Myths of Formal Methods," *IEEE Software* **7** (September 1990), pp. 11–19. (Chapter 11)

[Halstead, 1977] M. H. HALSTEAD, *Elements of Software Science,* Elsevier North-Holland, New York, 1977. (Chapters 9 and 14)

[Harel, 1992] D. HAREL, "Biting the Silver Bullet," *IEEE Computer* **25** (January 1992), pp. 8–24. (Chapter 2)

[Harel and Gery, 1997] D. HAREL AND E. GERY, "Executable Object Modeling with Statecharts," *IEEE Computer* **30** (July 1997), pp. 31–42. (Chapters 11 and 12)

[Harel et al., 1990] D. HAREL, H. LACHOVER, A. NAAMAD, A. PNUELI, M. POLITI, R. SHERMAN, A. SHTULL-TRAURING, AND M. TRAKHTENBROT, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering* **16** (April 1990), pp. 403–14. (Chapters 11 and 15)

[Harrold, McGregor, and Fitzpatrick, 1992] M. J. HARROLD, J. D. MCGREGOR, AND K. J. FITZPATRICK, "Incremental Testing of Object-Oriented Class Structures," *Proceedings of the 14th International Conference on Software Engineering,* Melbourne, Australia, May 1992, pp. 68–80. (Chapter 14)

[Harrold and Soffa, 1991] M. J. HARROLD AND M. L. SOFFA, "Selecting and Using Data for Integration Testing," *IEEE Software* **8** (1991), pp. 58–65. (Chapter 15)

[Henderson-Sellers, 1996] B. HENDERSON-SELLERS, *Object-Oriented Metrics: Measures of Complexity,* Prentice Hall, Upper Saddle River, NJ, 1996. (Chapters 5 and 7)

[Henderson-Sellers and Edwards, 1990] B. HENDERSON-SELLERS AND J. M. EDWARDS, "The Object-Oriented Systems Life Cycle," *Communications of the ACM* **33** (September 1990), pp. 142–59. (Chapter 3)

[Henry, Henry, Kafura, and Matheson, 1994] J. HENRY, S. HENRY, D. KAFURA, AND L. MATHESON, "Improving Software Maintenance at Martin Marietta," *IEEE Software* **11** (July 1994), pp. 67–75. (Chapter 16)

[Henry and Humphrey, 1990] S. M. HENRY AND M. HUMPHREY, "A Controlled Experiment to Evaluate Maintainability of Object-Oriented Software," *Proceedings of the IEEE Conference on Software Maintenance,* San Diego, CA, November 1990, pp. 258–65. (Chapter 16)

[Henry and Kafura, 1981] S. M. HENRY AND D. KAFURA, "Software Structure Metrics Based on Information Flow," *IEEE Transactions on Software Engineering* **SE-7** (September 1981), pp. 510–18. (Chapter 13)

[Henry and Selig, 1990] S. HENRY AND C. SELIG, "Predicting Source-Code Complexity at the Design Stage," *IEEE Software* **7** (March 1990), pp. 36–44. (Chapter 13)

[Herbsleb and Grinter, 1999] J. D. HERBSLEB AND R. E. GRINTER, "Architectures, Coordination, and Distance: Conway's Law and Beyond," *IEEE Software* **16** (September/October 1999), pp. 63–70. (Chapter 13)

[Herbsleb et al., 1994] J. HERBSLEB, A. CARLETON, J. ROZUM, J. SIEGEL, AND D. ZUBROW, "Benefits of CMM-Based Software Process Improvement: Initial Results," Technical Report CMU/SEI-94-TR-013, Software Engineering Institute, Carnegie Mellon University, August 1994. (Chapter 2)

[Herbsleb et al., 1997] J. HERBSLEB, D. ZUBROW, D. GOLDENSON, W. HAYES, AND M. PAULK, "Software Quality and the Capability Maturity

Model," *Communications of the ACM* **40** (June 1997), pp. 30–40. (Chapters 2 and 6)

[Hetzel, 1988] W. HETZEL, *The Complete Guide to Software Testing*, 2nd ed., QED Information Systems, Wellesley, MA, 1988. (Chapter 6)

[Hicks and Card, 1994] M. HICKS AND D. CARD, "Tales of Process Improvement," *IEEE Software* **11** (January 1994), pp. 114–15. (Chapter 2)

[Hoare, 1969] C. A. R. HOARE, "An Axiomatic Basis for Computer Programming," *Communications of the ACM* **12** (October 1969), pp. 576–83. (Chapter 6)

[Hoare, 1981] C. A. R. HOARE, "The Emperor's Old Clothes," *Communications of the ACM* **24** (February 1981), pp. 75–83. (Chapter 6)

[Hoare, 1985] C. A. R. HOARE, *Communicating Sequential Processes*, Prentice Hall International, Englewood Cliffs, NJ, 1985. (Chapter 11)

[Hoare, 1987] C. A. R. HOARE, "An Overview of Some Formal Methods for Program Design," *IEEE Computer* **20** (September 1987), pp. 85–91. (Chapter 13)

[Hofmeister, Nord, and Soni, 1999] C. HOFMEISTER, R. NORD, AND D. SONI, *Applied Software Architecture*, Addison-Wesley, Reading, MA, 1999. (Chapter 8)

[Holzner, 1993] S. HOLZNER, *Microsoft Foundation Class Library Programming*, Brady, New York, 1993. (Chapter 8)

[Honiden, Kotaka, and Kishimoto, 1993] S. HONIDEN, N. KOTAKA, AND Y. KISHIMOTO, "Formalizing Specification Modeling in OOA," *IEEE Software* **10** (January 1993), pp. 54–66. (Chapter 3)

[Horgan, London, and Lyu, 1994] J. R. HORGAN, S. LONDON, AND M. R. LYU, "Achieving Software Quality with Testing Coverage Measures," *IEEE Computer* **27** (September 1994), pp. 60–69. (Chapter 14)

[House and Newman, 1989] D. E. HOUSE AND W. F. NEWMAN, "Testing Large Software Products," *ACM SIGSOFT Software Engineering Notes* **14** (April 1989), pp. 71–78. (Chapter 15)

[Howden, 1987] W. E. HOWDEN, *Functional Program Testing and Analysis*, McGraw-Hill, New York, 1987. (Chapter 14)

[Humphrey, 1989] W. S. HUMPHREY, *Managing the Software Process*, Addison-Wesley, Reading, MA, 1989. (Chapter 2)

[Humphrey, 1996] W. S. HUMPHREY, "Using a Defined and Measured Personal Software Process," *IEEE Software* **13** (May 1996), pp. 77–88. (Chapter 2)

[Humphrey, 1999] W. S. HUMPHREY, "Pathways to Process Maturity: The Personal Software Process and Team Software Process," *SEI Interactive* **2** (No. 4, December 1999), http://interactive.sei.cmu.edu/Features/1999/June/Background/Background.jun99.htm. (Chapter 2)

[Humphrey, Snider, and Willis, 1991] W. S. HUMPHREY, T. R. SNIDER, AND R. R. WILLIS, "Software Process Improvement at Hughes Aircraft," *IEEE Software* **8** (July 1991), pp. 11–23. (Chapter 2)

[Hwang, 1981] S.-S. V. HWANG, "An Empirical Study in Functional Testing, Structural Testing, and Code Reading Inspection," Scholarly Paper 362, Department of Computer Science, University of Maryland, College Park, 1981. (Chapter 14)

[IEEE 610.12, 1990] "A Glossary of Software Engineering Terminology," IEEE 610.12-1990, Institute of Electrical and Electronic Engineers, New York, 1990. (Chapters 1, 6, and 16)

[IEEE 1028, 1997] "Standard for Software Reviews," IEEE 1028, Institute of Electrical and Electronic Engineers, New York, 1997. (Chapter 6)

[IEEE 1058.1, 1987] "Standard for Software Project Management Plans," IEEE 1058.1, Institute of Electrical and Electronic Engineers, New York, 1987. (Chapter 9)

[IEEE/EIA 12207, 1998] "IEEE/EIA 12207.0-1996 Industry Implementation of International Standard ISO/IEC 12207:1995," Institute of Electrical and Electronic Engineers, Electronic Industries Alliance, New York, 1998. (Chapter 2)

[Isakowitz and Kauffman, 1996] T. ISAKOWITZ AND R. J. KAUFFMAN, "Supporting Search for Reusable Software Objects," *IEEE Transactions on Software Engineering* **22** (June 1996), pp. 407–23. (Chapter 8)

[ISO 9000-3, 1991] "ISO 9000-3, Guidelines for the Application of ISO 9001 to the Development, Supply, and Maintenance of Software," International Organization for Standardization, Geneva, 1991. (Chapter 2)

[ISO 9001, 1987] "ISO 9001, Quality Systems—Model for Quality Assurance in Design/Development, Production, Installation, and Servicing," International Organization for Standardization, Geneva, 1987. (Chapter 2)

[ISO-7185, 1980] "Specification for the Computer Programming Language Pascal," ISO-7185, International Standards Organization, Geneva, 1980. (Chapter 8)

[ISO/IEC 1539–1, 1997] "Information Technology—Programming Languages—Fortran—Part 1: Base Language," ISO/IEC 1539–1, International Standards Organization, International Electrotechnical Commission, Geneva, 1997. (Chapter 8)

[ISO/IEC 1989, 2000] "Information Technology—Programming Languages, Their Environments and System Software Interfaces—Programming Language COBOL," Committee Draft 1.8, Proposed Revision of ISO 1989:1985, NCITS/J4 and ISO/IEC JCT1/SC22/WG4, International Organization for Standardization, International Electrotechnical Commission, Geneva, February 2000. (Chapter 8)

[ISO/IEC 8652, 1995] "Programming Language Ada: Language and Standard Libraries," ISO/IEC 8652, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995. (Chapters 7, 8, and 13)

[ISO/IEC 12207, 1995] "ISO/IEC 12207:1995, Information Technology—Software Life-Cycle Processes," International Organization for Standardization, International Electrotechnical Commission, Geneva, 1995. (Chapters 1, 2, 3, and 16)

[ISO/IEC 14882, 1998] "Programming Language C++," ISO/IEC 14882, International Organization for Standardization, International Electrotechnical Commission, Geneva, 1998. (Chapter 8)

[IWSSD, 1986] Call for Papers, Fourth International Workshop on Software Specification and Design, *ACM SIGSOFT Software Engineering Notes* **11** (April 1986), pp. 94–96. (Chapter 11)

[Jackson, 1975] M. A. JACKSON, *Principles of Program Design,* Academic Press, New York, 1975. (Chapters 11 and 13)

[Jackson, 1995] M. JACKSON, *Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices,* Addison Wesley Longman, Reading, MA, 1995. (Chapter 10)

[Jacobson, Booch, and Rumbaugh, 1999] I. JACOBSON, G. BOOCH, AND J. RUMBAUGH, *The Unified Software Development Process,* Addison-Wesley, Reading, MA, 1999. (Chapters 3, 12, 13, and 15)

[Jacobson, Christerson, Jonsson, and Overgaard, 1992] I. JACOBSON, M. CHRISTERSON, P. JONSSON, AND G. OVERGAARD, *Object-Oriented Software Engineering: A Use Case Driven Approach,* ACM Press, New York, 1992. (Chapter 12)

[Jazayeri, Ran, and van der Linden, 2000] M. JAZAYERI, A. RAN, AND F. VAN DER LINDEN, *Software Architecture for Product Families: Principles and Practice,* Addison-Wesley, Reading, MA, 2000. (Chapter 8)

[Jeffrey, Low, and Barnes, 1993] D. R. JEFFREY, G. C. LOW, AND M. BARNES, "A Comparison of Function Point Counting Techniques," *IEEE Transactions on Software Engineering* **19** (May 1993), pp. 529–32. (Chapter 9)

[Jensen and Wirth, 1975] K. JENSEN AND N. WIRTH, *Pascal User Manual and Report,* 2nd ed., Springer-Verlag, New York, 1975. (Chapter 8)

[Jézéquel and Meyer, 1997] J.-M. JÉZÉQUEL AND B. MEYER, "Put It in the Contract: The Lessons of Ariane," *IEEE Computer* **30** (January 1997), pp. 129–30. (Chapter 8)

[Johnson, 1979] S. C. JOHNSON, "A Tour through the Portable C Compiler," 7th ed., *UNIX Programmer's Manual,* Bell Laboratories, Murray Hill, NJ, January 1979. (Chapter 8)

[Johnson, 1997] R. E. JOHNSON, "Frameworks = (Components + Patterns)," *Communications of the ACM* **40** (October 1997), pp. 39–42. (Chapter 8)

[Johnson, 1998] P. M. JOHNSON, "Reengineering Inspection," *Communications of the ACM* **42** (February 1998), pp. 49–52. (Chapter 6)

[Johnson, 2000] R. A. JOHNSON, "The Ups and Downs of Object-Oriented System Development," *Communications of the ACM* **43** (October 2000), pp. 69–73. (Chapters 1 and 7)

[Johnson and Disney, 1998] P. M. JOHNSON AND A. M. DISNEY, "The Personal Software Process: A Cautionary Tale," *IEEE Software* **15** (November/December 1998), pp. 85–88. (Chapter 2)

[Johnson and Ritchie, 1978] S. C. JOHNSON AND D. M. RITCHIE, "Portability of C Programs and the UNIX System," *Bell System Technical Journal* **57** (No. 6, Part 2, 1978), pp. 2021–48. (Chapter 8)

[Jones, 1978] T. C. JONES, "Measuring Programming Quality and Productivity," *IBM Systems Journal* **17** (No. 1, 1978), pp. 39–63. (Chapter 6)

[Jones, 1984] T. C. JONES, "Reusability in Programming: A Survey of the State of the Art," *IEEE Transactions on Software Engineering* **SE-10** (September 1984), pp. 488–94. (Chapter 8)

[Jones, 1986a] C. JONES, *Programming Productivity,* McGraw-Hill, New York, 1986. (Chapter 9)

[Jones, 1986b] C. B. JONES, *Systematic Software Development Using VDM,* Prentice Hall, Englewood Cliffs, NJ, 1986. (Chapter 11)

[Jones, 1987] C. JONES, Letter to the Editor, *IEEE Computer* **20** (December 1987), p. 4. (Chapter 9)

[Jones, 1994a] C. JONES, "Software Metrics: Good, Bad, and Missing," *IEEE Computer* **27** (September 1994), pp. 98–100. (Chapter 5)

[Jones, 1994b] C. JONES, "Cutting the High Cost of Software 'Paperwork'," *IEEE Computer* **27** (October 1994), pp. 79–80. (Chapter 9)

[Jones, 1994c] C. JONES, *Assessment and Control of Computer Risks,* Prentice Hall, Englewood Cliffs, NJ, 1994. (Chapter 3)

[Jones, 1996] C. JONES, *Applied Software Measurement,* McGraw-Hill, New York, 1996. (Chapters 2 and 8)

[Joos, 1994] R. JOOS, "Software Reuse at Motorola," *IEEE Software* **11** (September 1994), pp. 42–47. (Chapter 8)

[Jorgensen and Erickson, 1994] P. C. JORGENSEN AND C. ERICKSON, "Object-Oriented Integration Testing," *Communications of the ACM* **37** (September 1994), pp. 30–38. (Chapter 15)

[Josephson, 1992] M. JOSEPHSON, *Edison: A Biography,* John Wiley and Sons, New York, 1992. (Chapter 1)

[Juran, 1988] J. M. JURAN, *Juran on Planning for Quality,* Macmillan, New York, 1988. (Chapter 2)

[Juristo, Moreno, and López, 2000] N. JURISTO, A. M. MORENO AND M. LÓPEZ, "How to Use Linguistic Instruments for Object-Oriented Analysis," *IEEE Software* **17** (May/June 2000), pp. 80–89. (Chapter 12)

[Kampen, 1987] G. R. KAMPEN, "An Eclectic Approach to Specification," *Proceedings of the Fourth International Workshop on Software Specification and Design,* Monterey, CA, April 1987, pp. 178–82. (Chapter 11)

[Kan et al., 1994] S. H. KAN, S. D. DULL, D. N. AMUNDSON, R. J. LINDNER, AND R. J. HEDGER, "AS/400 Software Quality Management," *IBM Systems Journal* **33** (No. 1, 1994), pp. 62–88. (Chapter 1)

[Karlsson and Ryan, 1997] J. KARLSSON AND K. RYAN, "A Cost-Value Approach for Prioritizing Requirements," *IEEE Software* **14** (September/October 1997), pp. 67–74. (Chapter 10)

[Karolak, 1996] D. W. KAROLAK, *Software Engineering Risk Management,* IEEE Computer Society, Los Alamitos, CA, 1996. (Chapter 3)

[Kazman, Abowd, Bass, and Clements, 1996] R. KAZMAN, G. ABOWD, L. BASS, AND P. CLEMENTS, "Scenario-Based Analysis of Software Architecture," *IEEE Software* **13** (November/December 1996), pp. 47–55. (Chapter 12)

[Keil, Cule, Lyytinen, and Schmidt, 1998] M. KEIL, P. E. CULE, K. LYYTINEN, AND R. C. SCHMIDT, "A Framework for Identifying Software Project Risks," *Communications of the ACM* **41** (November 1998), pp. 76–83. (Chapter 3)

[Kelly and Sherif, 1992] J. C. KELLY AND J. S. SHERIF, "A Comparison of Four Design Methods for Real-Time Software Development," *Information and Software Technology* **34** (February 1992), pp. 74–82. (Chapter 13)

[Kelly, Sherif, and Hops, 1992] J. C. KELLY, J. S. SHERIF, AND J. HOPS, "An Analysis of Defect Densities Found during Software Inspections," *Journal of Systems and Software* **17** (January 1992), pp. 111–17. (Chapters 1 and 6)

[Kemerer, 1993] C. F. KEMERER, "Reliability of Function Points Measurement: A Field Experiment," *Communications of the ACM* **36** (February 1993), pp. 85–97. (Chapter 9)

[Kemerer and Porter, 1992] C. F. KEMERER AND B. S. PORTER, "Improving the Reliability of Function Point Measurement: An Empirical Study," *IEEE Transactions on Software Engineering* **18** (November 1992), pp. 1011–24. (Chapter 9)

[Kernighan and Plauger, 1974] B. W. KERNIGHAN AND P. J. PLAUGER, *The Elements of Programming Style,* McGraw-Hill, New York, 1974. (Chapter 14)

[Kernighan and Ritchie, 1978] B. W. KERNIGHAN AND D. M. RITCHIE, *The C Programming Language,* Prentice Hall, Englewood Cliffs, NJ, 1978. (Chapter 8)

[Kerth and Cunningham, 1997] N. L. KERTH AND W. CUNNINGHAM, "Using Patterns to Improve Our Architectural Vision," *IEEE Software* **14** (January/February 1997), pp. 53–59. (Chapter 8)

[Khan, Al-A'ali, and Girgis, 1995] E. H. KHAN, M. AL-A'ALI, AND M. R. GIRGIS, "Object-Oriented Programming for Structured Procedural Programming," *IEEE Computer* **28** (October 1995), pp. 48–57. (Chapter 1)

[King and Schrems, 1978] J. L. KING AND E. L. SCHREMS, "Cost–Benefit Analysis in Information Systems Development and Operation," *ACM Computer Surveys* **10** (March 1978), pp. 19–34. (Chapter 5)

[Kitchenham, 1997] B. KITCHENHAM, "The Problem with Function Points," IEEE Software 14 (March/April 1997), pp. 29, 31. (Chapter 9)

[Kitchenham and Känsälä, 1993] B. KITCHENHAM AND K. KÄNSÄLÄ, "Inter-Item Correlations among Function Points," *Proceedings of the IEEE 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 477–80. (Chapter 9)

[Kitchenham, Pickard, and Linkman, 1990] B. A. KITCHENHAM, L. M. PICKARD, AND S. J. LINKMAN, "An Evaluation of Some Design Metrics," *Software Engineering Journal* **5** (January 1990), pp. 50–58. (Chapter 13)

[Kitchenham, Pickard, and Pfleeger, 1995] B. KITCHENHAM, L. PICKARD, AND S. L. PFLEEGER, "Case Studies for Method and Tool Evaluation," *IEEE Software* **12** (July 1995), pp. 52–62. (Chapter 5)

[Kitson and Masters, 1993] D. H. KITSON AND S. M. MASTERS, "An Analysis of SEI Software Process Assessment Results: 1987–1991," *Proceedings of the IEEE 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 68–77. (Chapter 2)

[Klatte et al., 1993] R. KLATTE, U. KULISCH, A. WIETHOFF, C. LAWO, AND M. RAUCH, *C-XSC: A C++ Class Library for Extended Scientific Computing,* Springer-Verlag, Heidelberg, Germany, 1993. (Chapter 8)

[Kleinrock and Gail, 1996] L. KLEINROCK AND R. GAIL, *Queuing Systems: Problems and Solutions,* John Wiley and Sons, New York, 1996. (Chapter 11)

[Klepper and Bock, 1995] R. KLEPPER AND D. BOCK, "Third and Fourth Generation Productivity Differences," *Communications of the ACM* **38** (September 1995), pp. 69–79. (Chapter 14)

[Klunder, 1988] D. KLUNDER, "Hungarian Naming Conventions," Technical Report, Microsoft Corporation, Redmond, WA, January 1988. (Chapter 14)

[Knuth, 1968] D. E. KNUTH, *The Art of Computer Programming,* Volume I, *Fundamental Algorithms,* Addison-Wesley, Reading, MA, 1968. (Chapter 11)

[Knuth, 1974] D. E. KNUTH, "Structured Programming with **go to** Statements," *ACM Computing Surveys* **6** (December 1974), pp. 261–301. (Chapter 7)

[Kobryn, 2000] C. KOBRYN, "Modeling Components and Frameworks with UML," *Communications of the ACM* **43** (October 2000), pp. 31–38. (Chapter 8)

[Kramer, 1994] J. KRAMER, "Distributed Software Engineering," *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 253–63. (Chapters 11 and 13)

[Kroeker et al., 1999] K. K. KROEKER, L. WALL, D. A. TAYLOR, C. HORN, P. BASSETT, J. K. OUSTERHOUT, M. L. GRISS, R. M. SOLEY, J. WALDO, AND C. SIMONYI, "Software [R]evolution: A Roundtable," *IEEE Computer* **32** (May 1999), pp. 48–57. (Chapter 1)

[Kung, Hsia, and Gao, 1998] D. C. KUNG, P. HSIA, AND J. GAO, *Testing Object-Oriented Software,* IEEE Computer Society Press, Los Alamitos, CA, 1998. (Chapter 6)

[Kurki-Suonio, 1993] R. KURKI-SUONIO, "Stepwise Design of Real-Time Systems," *IEEE Transactions on Software Engineering* **19** (January 1993), pp. 56–69. (Chapter 5)

[Lai, Weiss, and Parnas, 1999] C. T. R. LAI, D. M. WEISS, AND D. L. PARNAS, *Software Product-Line Engineering: A Family-Based Software Development Process,* Addison-Wesley, Reading, MA, 1999. (Chapter 8)

[Landis et al., 1992] L. LANDIS, S. WALIGARA, F. MCGARRY, ET AL., "Recommended Approach to Software Development: Revision 3," Technical Report SEL-81-305, Software Engineering Laboratory, Greenbelt, MD, June 1992. (Chapter 3)

[Landwehr, 1983] C. E. LANDWEHR, "The Best Available Technologies for Computer Security," *IEEE Computer* **16** (July 1983), pp. 86–100. (Chapter 6)

[Lanergan and Grasso, 1984] R. G. LANERGAN AND C. A. GRASSO, "Software Engineering with Reusable Designs and Code," *IEEE Transactions on Software Engineering* **SE-10** (September 1984), pp. 498–501. (Chapter 8)

[Langtangen, 1994] H. P. LANGTANGEN, "Getting Started with Finite Element Programming in DIFFPACK," The DiffPack Report Series, SINTEF, Oslo, Norway, October 2, 1994. (Chapter 8)

[Larsen, Fitzgerald, and Brookes, 1996] P. G. LARSEN, J. FITZGERALD, AND T. BROOKES, "Applying Formal Specification in Industry," *IEEE Software* **13** (May 1996), pp. 48–56. (Chapter 11)

[Leavenworth, 1970] B. LEAVENWORTH, Review #19420, *Computing Reviews* **11** (July 1970), pp. 396–97. (Chapters 6 and 11)

[Lederer and Prasad, 1992] A. L. LEDERER AND J. PRASAD, "Nine Management Guidelines for Better Cost Estimating," *Communications of the ACM* **35** (February 1992), pp. 51–59. (Chapter 9)

[Lee and Tepfenhart, 1997] R. C. LEE AND W. M. TEPFENHART, *UML: A Practical Guide to Object-Oriented Development,* Prentice Hall, Upper Saddle River, NJ, 1997. (Chapter 12)

[Lejter, Meyers, and Reiss, 1992] M. LEJTER, S. MEYERS, AND S. P. REISS, "Support for Maintaining Object-Oriented Programs," *IEEE Transactions on Software Engineering* **18** (December 1992), pp. 1045–52. (Chapter 16)

[Leppinen, Pulkkinen, and Rautiainen, 1997] M. LEPPINEN, P. PULKKINEN, AND A. RAUTIAINEN, "JAVA- and CORBA-Based Network Management," *IEEE Computer* **30** (June 1997), pp. 83–87. (Chapter 8)

[Leveson, 1997] N. G. LEVESON, "Software Engineering: Stretching the Limits of Complexity," *Communications of the ACM* **40** (February 1997), pp. 129–31. (Chapter 1)

[Leveson and Turner, 1993] N. G. LEVESON AND C. S. TURNER, "An Investigation of the Therac-25 Accidents," *IEEE Computer* **26** (July 1993), pp. 18–41. (Chapter 1)

[Levi and Agrawala, 1990] S.-T. LEVI AND A. K. AGRAWALA, *Real Time System Design*, McGraw-Hill, New York, 1990. (Chapter 13)

[Lewis, 1996a] T. LEWIS, "The Next $10{,}000_2$ Years: Part I," *IEEE Computer* **29** (April 1996), pp. 64–70. (Chapter 1)

[Lewis, 1996b] T. LEWIS, "The Next $10{,}000_2$ Years: Part II," *IEEE Computer* **29** (May 1996), pp. 78–86. (Chapter 1)

[Lewis et al., 1995] T. LEWIS, L. ROSENSTEIN, W. PREE, A. WEINAND, E. GAMMA, P. CALDER, G. ANDERT, J. VLISSIDES, AND K. SCHMUCKER, *Object-Oriented Application Frameworks*, Manning, Greenwich, CT, 1995. (Chapter 8)

[Lichter, Schneider-Hufschmidt, and Züllighoven, 1994] H. LICHTER, M. SCHNEIDER-HUFSCHMIDT, AND H. ZÜLLIGHOVEN, "Prototyping in Industrial Software Projects—Bridging the Gap between Theory and Practice," *IEEE Transactions on Software Engineering* **20** (November 1994), pp. 825–32. (Chapter 10)

[Lientz and Swanson, 1980] B. P. LIENTZ AND E. B. SWANSON, *Software Maintenance Management: A Study of the Maintenance of Computer Applications Software in 487 Data Processing Organizations*, Addison-Wesley, Reading, MA, 1980. (Chapter 16)

[Lientz, Swanson, and Tompkins, 1978] B. P. LIENTZ, E. B. SWANSON, AND G. E. TOMPKINS, "Characteristics of Application Software Maintenance," *Communications of the ACM* **21** (June 1978), pp. 466–71. (Chapters 1 and 16)

[Lim, 1994] W. C. LIM, "Effects of Reuse on Quality, Productivity, and Economics," *IEEE Software* **11** (September 1994). pp. 23–30. (Chapter 8 and 9)

[Lim, 1998] W. C. LIM, *Managing Software Reuse*, Prentice Hall, Upper Saddle River, NJ, 1998. (Chapter 8)

[Linger, 1994] R. C. LINGER, "Cleanroom Process Model," *IEEE Software* **11** (March 1994), pp. 50–58. (Chapter 14)

[Liskov and Zilles, 1974] B. LISKOV AND S. ZILLES, "Programming with Abstract Data Types," *ACM SIGPLAN Notices* **9** (April 1974), pp. 50–59. (Chapter 7)

[Liskov, Snyder, Atkinson, and Schaffert, 1977] B. LISKOV, A. SNYDER, R. ATKINSON, AND C. SCHAFFERT, "Abstraction Mechanisms in CLU," *Communications of the ACM* **20** (August 1977), pp. 564–76. (Chapter 8)

[Littlewood and Strigini, 1992] B. LITTLEWOOD AND L. STRIGINI, "The Risks of Software," *Scientific American* **267** (November 1992), pp. 62–75. (Chapter 1)

[London, 1971] R. L. LONDON, "Software Reliability through Proving Programs Correct," *Proceedings of the IEEE International Symposium on Fault-Tolerant Computing*, March 1971. (Chapters 6 and 11)

[Long and Morris, 1993] F. LONG AND E. MORRIS, "An Overview of PCTE: A Basis for a Portable Common Tool Environment," Technical Report CMU/SEI–93–TR–1, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, January 1993. (Chapter 15)

[Longstreet, 1990] D. H. LONGSTREET (EDITOR), *Software Maintenance and Computers*, IEEE Computer Society Press, Los Alamitos, CA, 1990. (Chapter 16)

[Loukides and Oram, 1997] M. K. LOUKIDES AND A. ORAM, *Programming with GNU Software*, O'Reilly and Associates, Sebastopol, CA, 1997. (Chapter 5, 15, and 16)

[Low and Jeffrey, 1990] G. C. LOW AND D. R. JEFFREY, "Function Points in the Estimation and Evaluation of the Software Process," *IEEE Transactions on Software Engineering* **16** (January 1990), pp. 64–71. (Chapter 9)

[Luckham and von Henke, 1985] D. C. LUCKHAM AND F. W. VON HENKE, "An Overview of Anna, a Specification Language for Ada," *IEEE Software* **2** (March 1985), pp. 9–22. (Chapter 11)

[Luqi and Royce, 1992] LUQI AND W. ROYCE, "Status Report: Computer-Aided Prototyping," *IEEE Software* **9** (November 1992), pp. 77–81. (Chapter 3)

[Mackenzie, 1980] C. E. MACKENZIE, *Coded Character Sets: History and Development,* Addison-Wesley, Reading, MA, 1980. (Chapter 8)

[Mackey, 1999] K. MACKEY, "Stages of Team Development," *IEEE Software* **16** (July/August 1999), pp. 90–91. (Chapter 4)

[Mancl and Havanas, 1990] D. MANCL AND W. HAVANAS, "A Study of the Impact of C++ on Software Maintenance," *Proceedings of the IEEE Conference on Software Maintenance,* San Diego, CA, November 1990, pp. 63–69. (Chapter 16)

[Manna and Pnueli, 1992] Z. MANNA AND A. PNUELI, *The Temporal Logic of Reactive and Concurrent Systems,* Springer-Verlag, New York, 1992. (Chapter 6)

[Manna and Waldinger, 1978] Z. MANNA AND R. WALDINGER, "The Logic of Computer Programming," *IEEE Transactions on Software Engineering* **SE-4** (1978), pp. 199–229. (Chapter 6)

[Mantei, 1981] M. MANTEI, "The Effect of Programming Team Structures on Programming Tasks," *Communications of the ACM* **24** (March 1981), pp. 106–13. (Chapter 4)

[Mantei and Teorey, 1988] M. M. MANTEI AND T. J. TEOREY, "Cost/Benefit Analysis for Incorporating Human Factors in the Software Development Lifecycle," *Communications of the ACM* **31** (April 1988), pp. 428–39. (Chapter 3)

[Maring, 1996] B. MARING, "Object-Oriented Development of Large Applications," *IEEE Software* **13** (May 1996), pp. 33–40. (Chapter 1)

[Martin, 1985] J. MARTIN, *Fourth-Generation Languages,* Volumes 1, 2, and 3, Prentice Hall, Englewood Cliffs, NJ, 1985. (Chapter 14)

[Matsumoto, 1984] Y. MATSUMOTO, "Management of Industrial Software Production," *IEEE Computer* **17** (February 1984), pp. 59–72. (Chapter 8)

[Matsumoto, 1987] Y. MATSUMOTO, "A Software Factory: An Overall Approach to Software Production," in: *Tutorial: Software Reusability,* P. Freeman (Editor), Computer Society Press, Washington, DC, 1987, pp. 155–78. (Chapter 8)

[Maxwell and Forselius, 2000] K. D. MAXWELL AND P. FORSELIUS, "Benchmarking Software Development Productivity," *IEEE Software* **17** (January/February 2000), pp. 80–88. (Chapter 9)

[Mays, 1994] R. G. MAYS, "Forging a Silver Bullet from the Essence of Software," *IBM Systems Journal* **33** (No. 1, 1994) pp. 20–45. (Chapter 2)

[McCabe, 1976] T. J. MCCABE, "A Complexity Measure," *IEEE Transactions on Software Engineering* **SE-2** (December 1976), pp. 308–20. (Chapters 13 and 14)

[McCabe and Butler, 1989] T. J. MCCABE AND C. W. BUTLER, "Design Complexity Measurement and Testing," *Communications of the ACM* **32** (December 1989), pp. 1415–25. (Chapter 14)

[McConnell, 1993] S. MCCONNELL, *Code Complete: A Practical Handbook of Software Construction,* Microsoft Press, Redmond, WA, 1993. (Chapter 14)

[McConnell, 1996] S. MCCONNELL, "Daily Build and Smoke Test," *IEEE Computer* **13** (July 1996), pp. 144, 143. (Chapters 3 and 4)

[Mellor, 1994] P. MELLOR, "CAD: Computer-Aided Disaster," Technical Report, Centre for Software Reliability, City University, London, U.K., July 1994. (Chapter 1)

[Meyer, 1985] B. MEYER, "On Formalism in Specifications," *IEEE Software* **2** (January 1985), pp. 6–26. (Chapter 11)

[Meyer, 1986] B. MEYER, "Genericity versus Inheritance," Proceedings of the Conference on Object-Oriented Programming Systems, Languages and Applications, *ACM SIGPLAN Notices* **21** (November 1986), pp. 391–405. (Chapter 7)

[Meyer, 1987] B. MEYER, "Reusability: The Case for Object-Oriented Design," *IEEE Software* **4** (March 1987), pp. 50–64. (Chapter 8)

[Meyer, 1990] B. MEYER, "Lessons from the Design of the Eiffel Libraries," *Communications of the ACM* **33** (September 1990), pp. 68–88. (Chapters 7, 8, and 16)

[Meyer, 1992a] B. MEYER, "Applying 'Design by Contract'," *IEEE Computer* **25** (October 1992), pp. 40–51. (Chapters 1 and 7)

[Meyer, 1992b] B. MEYER, *Eiffel: The Language,* Prentice Hall, New York, 1992. (Chapters 6, 7, and 13)

[Meyer, 1996a] B. MEYER, "The Reusability Challenge," *IEEE Computer* **29** (February 1996), pp. 76–78. (Chapter 8)

[Meyer, 1996b] B. MEYER, "The Many Faces of Inheritance: A Taxonomy of Taxonomy," *IEEE Computer* **29** (May 1996), pp. 105–8. (Chapter 7)

[Meyer, 1997] B. MEYER, *Object-Oriented Software Construction,* 2nd ed., Prentice Hall, Upper Saddle River, NJ, 1997. (Chapters 1 and 7)

[Mili, Addy, Mili, and Yacoub, 1999] A. MILI, E. ADDY, H. MILI, AND S. YACOUB, "Toward an Engineering Discipline of Software Reuse," *IEEE Software* **16** (September/October 1999), pp. 22–31. (Chapter 8)

[Miller, 1956] G. A. MILLER, "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information," *The Psychological Review* **63** (March 1956), pp. 81–97. (Chapter 5)

[Mills, 1988] H. D. MILLS, "Stepwise Refinement and Verification in Box-Structured Systems," *IEEE Computer* **21** (June 1988), pp. 23–36. (Chapter 5)

[Mills, Dyer, and Linger, 1987] H. D. MILLS, M. DYER, AND R. C. LINGER, "Cleanroom Software Engineering," *IEEE Software* **4** (September 1987), pp. 19–25. (Chapter 14)

[Mills, Linger, and Hevner, 1987] H. D. MILLS, R. C. LINGER, AND A. R. HEVNER, "Box Structured Information Systems," *IBM Systems Journal* **26** (No. 4, 1987), pp. 395–413. (Chapter 5)

[Modell, 1996] M. E. MODELL, *A Professional's Guide to Systems Analysis,* 2nd ed., McGraw-Hill, 1996. (Chapter 11)

[Monarchi and Puhr, 1992] D. E. MONARCHI AND G. I. PUHR, "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM* **35** (September 1992), pp. 35–47. (Chapters 12 and 13)

[Monroe, Kompanek, Melton, and Garlan, 1997] R. T. MONROE, A. KOMPANEK, R. MELTON, AND D. GARLAN, "Architectural Styles, Design Patterns, and Objects," *IEEE Software* **14** (January/February 1997), pp. 43–52. (Chapter 8)

[Mooney, 1990] J. D. MOONEY, "Strategies for Supporting Application Portability," *IEEE Computer* **23** (November 1990), pp. 59–70. (Chapter 8)

[Moran, 1981] T. P. MORAN (EDITOR), Special Issue: The Psychology of Human-Computer Interaction, *ACM Computing Surveys* **13** (March 1981). (Chapter 5)

[Moser and Nierstrasz, 1996] S. MOSER AND O. NIERSTRASZ, "The Effect of Object-Oriented Frameworks on Developer Productivity," *IEEE Computer* **29** (September 1996), pp. 45–51. (Chapters 8 and 9)

[Mowbray and Zahavi, 1995] T. J. MOWBRAY AND R. ZAHAVI, *The Essential CORBA,* John Wiley and Sons, New York, 1995. (Chapter 8)

[Munoz, 1988] C. U. MUNOZ, "An Approach to Software Product Testing," *IEEE Transactions on Software Engineering* **14** (November 1988), pp. 1589–96. (Chapter 15)

[Musa and Everett, 1990] J. D. MUSA AND W. W. EVERETT, "Software-Reliability Engineering: Technology for the 1990s," *IEEE Software* **7** (November 1990), pp. 36–43. (Chapter 14)

[Musa, Iannino, and Okumoto, 1987] J. D. MUSA, A. IANNINO, AND K. OKUMOTO, *Software Reliability: Measurement, Prediction, Application,* McGraw-Hill, New York, 1987. (Chapter 14)

[Musser and Saini, 1996] D. R. MUSSER AND A. SAINI, *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library,* Addison-Wesley, Reading, MA, 1996. (Chapter 8)

[Myers, 1976] G. J. MYERS, *Software Reliability: Principles and Practices,* Wiley-Interscience, New York, 1976. (Chapter 14)

[Myers, 1978a] G. J. MYERS, "A Controlled Experiment in Program Testing and Code Walkthroughs/ Inspections," *Communications of the ACM* **21** (September 1978), pp. 760–68. (Chapter 14)

[Myers, 1978b] G. J. MYERS, *Composite/Structured Design,* Van Nostrand Reinhold, New York, 1978. (Chapter 7)

[Myers, 1979] G. J. MYERS, *The Art of Software Testing,* John Wiley and Sons, New York, 1979. (Chapters 6, 14, and 15)

[Myers, 1989] W. MYERS, "Allow Plenty of Time for Large-Scale Software," *IEEE Software* **6** (July 1989), pp. 92–99. (Chapter 9)

[Myers, 1992] W. MYERS, "Good Software Practices Pay Off—or Do They?" *IEEE Software* **9** (March 1992), pp. 96–97. (Chapter 5)

[Myers and Rosson, 1992] B. A. MYERS AND M. B. ROSSON, "Survey on User Interface Programming," *Proceedings of ACM SIGCHI Conference on Human Factors in Computing Systems,* Monterey, CA, May 1992, pp. 195–202. (Chapter 10)

[Naur, 1964] P. NAUR, "The Design of the GIER ALGOL Compiler," in: *Annual Review in Automatic Programming,* Volume 4, Pergamon Press, Oxford, U.K., 1964, pp. 49–85. (Chapter 11)

[Naur, 1969] P. NAUR, "Programming by Action Clusters,"*BIT* **9** (No. 3, 1969), pp. 250–58. (Chapters 6 and 11)

[Naur, Randell, and Buxton, 1976] P. NAUR, B. RANDELL, AND J. N. BUXTON (EDITORS),

Software Engineering: Concepts and Techniques: Proceedings of the NATO Conferences, Petrocelli-Charter, New York, 1976. (Chapter 1)

[Nesi, 1998] P. NESI, "Managing OO Projects Better," *IEEE Software* **15** (July/August 1998), pp. 50–60. (Chapter 9)

[Neumann, 1980] P. G. NEUMANN, Letter from the Editor, *ACM SIGSOFT Software Engineering Notes* **5** (July 1980), p. 2. (Chapter 1)

[Neumann, 1995] P. G. NEUMANN, *Computer-Related Risks,* Addison-Wesley, Reading, MA, 1995. (Chapter 1)

[New, 1992] R. NEW, personal communication, 1992. (Chapter 6)

[Nielsen, 1993] J. NIELSEN, "Iterative User-Interface Design," *IEEE Computer* **26** (November 1993), pp. 32–41. (Chapter 10)

[Nissen and Wallis, 1984] J. NISSEN AND P. WALLIS (EDITORS), *Portability and Style in Ada,* Cambridge University Press, Cambridge, U.K., 1984. (Chapter 8)

[NIST 151, 1988] "POSIX: Portable Operating System Interface for Computer Environments," Federal Information Processing Standard 151, National Institute of Standards and Technology, Washington, DC, 1988. (Chapter 8)

[Nix and Collins, 1988] C. J. NIX AND B. P. COLLINS, "The Use of Software Engineering, Including the Z Notation, in the Development of CICS," *Quality Assurance* **14** (September 1988), pp. 103–10. (Chapter 11)

[Norden, 1958] P. V. NORDEN, "Curve Fitting for a Model of Applied Research and Development Scheduling," *IBM Journal of Research and Development* **2** (July 1958), pp. 232–48. (Chapter 9)

[Norusis, 2000] M. J. NORUSIS, *SPSS 10.0 Guide to Data Analysis,* Prentice Hall, Upper Saddle Valley River, NJ, 2000. (Chapter 8)

[Oest, 1986] O. N. OEST, "VDM from Research to Practice," *Proceedings of the IFIP Congress, Information Processing '86,* 1986, pp. 527–33. (Chapter 11)

[OMG, 1999] "The Common Object Request Broker: Architecture and Specification. Revision 2.3.1," Document 99.10.07, Object Management Group, Needham, MA, October 1999. (Chapter 8)

[Onoma and Yamaura, 1995] A. K. ONOMA AND T. YAMAURA, "Practical Steps toward Quality Development," *IEEE Software* **12** (September 1995), pp. 68–77. (Chapter 6)

[Onoma, Tsai, Poonawala, and Suganuma, 1998] A. K. ONOMA, W.-T. TSAI, M. H. POONAWALA, AND H. SUGANUMA, "Regression Testing in an Industrial Environment," *Communications of the ACM* **42** (May 1998), pp. 81–86. (Chapter 16)

[Orfali, Harkey, and Edwards, 1996] R. ORFALI, D. HARKEY, AND J. EDWARDS, *The Essential Distributed Objects Survival Guide,* John Wiley and Sons, New York, 1996. (Chapter 8)

[Orr, 1981] K. ORR, *Structured Requirements Definition,* Ken Orr and Associates, Inc., Topeka, KS, 1981. (Chapters 11 and 13)

[Ottenstein, 1979] L. M. OTTENSTEIN, "Quantitative Estimates of Debugging Requirements," *IEEE Transactions on Software Engineering* **SE-5** (September 1979), pp. 504–14. (Chapter 14)

[Parnas, 1971] D. L. PARNAS, "Information Distribution Aspects of Design Methodology," *Proceedings of the IFIP Congress,* Ljubljana, Yugoslavia, 1971, pp. 339–44. (Chapter 7)

[Parnas, 1972a] D. L. PARNAS, "A Technique for Software Module Specification with Examples," *Communications of the ACM* **15** (May 1972), pp. 330–36. (Chapter 7)

[Parnas, 1972b] D. L. PARNAS, "On the Criteria to Be Used in Decomposing Systems into Modules," *Communications of the ACM* **15** (December 1972), pp. 1053–58. (Chapter 7)

[Parnas, 1979] D. L. PARNAS, "Designing Software for Ease of Extension and Contraction," *IEEE Transactions on Software Engineering* **SE-5** (March 1979), pp. 128–38. (Chapter 2)

[Parnas, 1990] D. L. PARNAS, "Education for Computing Professionals," *IEEE Computer* **23** (January 1990), pp. 17–22. (Chapter 1)

[Parnas, 1994] D. L. PARNAS, "Software Aging," *Proceedings of the 16th International Conference on Software Engineering,* Sorrento, Italy, May 1994, pp. 279–87. (Chapter 1)

[Parnas, 1999] D. L. PARNAS, "Ten Myths about Y2K Inspections," *Communications of the ACM* **42** (May 1999), p. 128. (Chapter 16)

[Paulk, 1995] M. C. PAULK, "How ISO 9001 Compares with the CMM," *IEEE Software* **12** (January 1995), pp. 74–83. (Chapter 2)

[Paulk, Weber, Curtis, and Chrissis, 1995] M. C. PAULK, C. V. WEBER, B. CURTIS, AND M. B. CHRISSIS, *The*

Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, Reading, MA, 1995. (Chapter 2)

[Perry and Kaiser, 1990] D. E. PERRY AND G. E. KAISER, "Adequate Testing and Object-Oriented Programming," *Journal of Object-Oriented Programming* **2** (January/February 1990), pp. 13–19. (Chapter 14)

[Peterson, 1981] J. L. PETERSON, *Petri Net Theory and the Modeling of Systems,* Prentice Hall, Englewood Cliffs, NJ, 1981. (Chapter 11)

[Petri, 1962] C. A. PETRI, "Kommunikation mit Automaten," Ph.D. Dissertation, University of Bonn, Germany, 1962. [In German.] (Chapter 11)

[Pfleeger and Hatton, 1997] S. L. PFLEEGER AND L. HATTON, "Investigating the Influence of Formal Methods," *IEEE Computer* **30** (February 1997), pp. 33–43. (Chapter 11)

[Pfleeger, Jeffrey, Curtis, and Kitchenham, 1997] S. L. PFLEEGER, R. JEFFREY, B. CURTIS, AND B. KITCHENHAM, "Status Report on Software Measurement," *IEEE Software* **14** (March/April 1997), pp. 33–44. (Chapter 5)

[Phillips, 1986] J. PHILLIPS, *The NAG Library: A Beginner's Guide,* Clarendon Press, Oxford, U.K., 1986. (Chapter 8)

[Pigoski, 1996] T. M. PIGOSKI, *Practical Software Maintenance: Best Practices for Managing Your Software Investment,* John Wiley and Sons, New York, 1996. (Chapter 16)

[Pittman, 1993] M. PITTMAN, "Lessons Learned in Managing Object-Oriented Development," *IEEE Software* **10** (January 1993), pp. 43–53. (Chapter 9)

[Pollack, Hicks, and Harrison, 1971] S. L. POLLACK, H. T. HICKS, JR., AND W. J. HARRISON, *Decision Tables: Theory and Practice,* Wiley-Interscience, New York, 1971. (Chapter 11)

[Ponder and Bush, 1994] C. PONDER AND B. BUSH, "Polymorphism Considered Harmful," *ACM SIGSOFT Software Engineering Notes* **19** (April 1994), pp. 35–38. (Chapter 7)

[Porter, Siy, Toman, and Votta, 1997] A. A. PORTER, H. P. SIY, C. A. TOMAN, AND L. G. VOTTA, "Assessing Software Review Meetings: Results of a Comparative Analysis of Two Experimental Studies," *IEEE Transactions on Software Engineering* **23** (March 1997), pp. 129–45. (Chapter 6)

[Poulin, 1997] J. S. POULIN, *Measuring Software Reuse: Principles, Practice, and Economic Models,* Addison-Wesley, Reading, MA, 1997. (Chapter 8)

[Preece, 1994] J. PREECE, *Human-Computer Interaction,* Addison-Wesley, Reading, MA, 1994. (Chapter 10)

[Prieto-Díaz, 1991] R. PRIETO-DÍAZ, "Implementing Faceted Classification for Software Reuse," *Communications of the ACM* **34** (May 1991), pp. 88–97. (Chapter 8)

[Prieto-Díaz, 1993] R. PRIETO-DÍAZ, "Status Report: Software Reusability," *IEEE Software* **10** (May 1993), pp. 61–66. (Chapter 8)

[Putnam, 1978] L. H. PUTNAM, "A General Empirical Solution to the Macro Software Sizing and Estimating Problem," *IEEE Transactions on Software Engineering* **SE-4** (July 1978), pp. 345–61. (Chapter 9)

[Radin, 1996] G. RADIN, "Object Technology in Perspective," *IBM Systems Journal* **35** (No. 2, 1996), pp. 124–126. (Chapter 1)

[Rajlich, 1985] V. RAJLICH, "Stepwise Refinement Revisited," *Journal of Systems and Software* **5** (February 1985), pp. 81–88. (Chapter 5)

[Rajlich, 1994] V. RAJLICH, "Decomposition/Generalization Methodology for Object-Oriented Programming," *Journal of Systems and Software* **24** (February 1994), pp. 181–86. (Chapter 3)

[Rajlich and Bennett, 2000] V. RAJLICH AND K. H. BENNETT, "A Staged Model for the Software Life Cycle," *IEEE Computer* **33** (July 2000), pp. 66–71. (Chapter 3)

[Rapps and Weyuker, 1985] S. RAPPS AND E. J. WEYUKER, "Selecting Software Test Data Using Data Flow Information," *IEEE Transactions on Software Engineering* **SE-11** (April 1985), pp. 367–75. (Chapter 14)

[Reel, 1999] J. S. REEL, "Critical Success Factors in Software Projects," *IEEE Software* **16** (May/June 1999), pp. 18–23. (Chapter 1)

[Reifer, 2000] D. J. REIFER, "Software Management: The Good, the Bad, and the Ugly," *IEEE Software* **17** (March/April 2000), pp. 73–75. (Chapter 9)

[Rettig, 1994] M. RETTIG, "Prototyping for Tiny Fingers," *Communications of the ACM* **37** (April 1994), pp. 21–27. (Chapter 10)

[Rochkind, 1975] M. J. ROCHKIND, "The Source Code Control System," *IEEE Transactions on Software Engineering* **SE-1** (October 1975), pp. 255–65. (Chapters 5, 15, and 16)

[Rosenblum and Weyuker, 1997] D. S. ROSENBLUM AND E. J. WEYUKER, "Using Coverage Information to Predict the Cost-Effectiveness of Regression Testing Strategies," *IEEE Transactions on Software Engineering* **23** (March 1997), pp. 146–56. (Chapter 16)

[Ross, 1985] D. T. ROSS, "Applications and Extensions of SADT," *IEEE Computer* **18** (April 1985), pp. 25–34. (Chapter 11)

[Rothermel and Harrold, 1996] G. ROTHERMEL AND M. J. HARROLD, "Analyzing Regression Test Selection Techniques," *IEEE Transactions on Software Engineering* **22** (August 1996), pp. 529–51. (Chapter 16)

[Royce, 1970] W. W. ROYCE, "Managing the Development of Large Software Systems: Concepts and Techniques," *1970 WESCON Technical Papers, Western Electronic Show and Convention,* Los Angeles, August 1970, pp. A/1-1–A/1-9. Reprinted in *Proceedings of the 11th International Conference on Software Engineering,* Pittsburgh, May 1989, pp. 328–38. (Chapter 3)

[Royce, 1998] W. ROYCE, *Software Project Management: A Unified Framework,* Addison-Wesley, Reading, MA, 1998. (Chapters 3 and 4)

[Rugaber and White, 1998] S. RUGABER AND J. WHITE, "Restoring a Legacy: Lessons Learned," *IEEE Software* **15** (July/August 1998), pp. 28–33. (Chapter 16)

[Rumbaugh, Jacobson, and Booch, 1999] J. RUMBAUGH, I. JACOBSON, AND G. BOOCH, *The Unified Modeling Language Reference Manual,* Addison-Wesley, Reading, MA, 1999. (Chapter 12)

[Rumbaugh et al., 1991] J. RUMBAUGH, M. BLAHA, W. PREMERLANI, F. EDDY, AND W. LORENSEN, *Object-Oriented Modeling and Design,* Prentice Hall, Englewood Cliffs, NJ, 1991. (Chapter 12)

[Russell, 1991] G. W. RUSSELL, "Experience with Inspection in Ultralarge-Scale Developments," *IEEE Software* **8** (January 1991), pp. 25–31. (Chapter 6)

[Sackman, 1970] H. SACKMAN, *Man–Computer Problem Solving: Experimental Evaluation of Time-Sharing and Batch Processing,* Auerbach, Princeton, NJ, 1970. (Chapter 9)

[Sackman, Erikson, and Grant, 1968] H. SACKMAN, W. J. ERIKSON, AND E. E. GRANT, "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM* **11** (January 1968), pp. 3–11. (Chapter 9)

[Saiedian and Kuzara, 1995] H. SAIEDIAN AND R. KUZARA, "SEI Capability Maturity Model's Impact on Contractors," *IEEE Computer* **28** (January 1995), pp. 16–26. (Chapter 2)

[Saiedian et. al., 1996] H. SAIEDIAN, J. P. BOWEN, R. W. BUTLER, D. L. DILL, R. L. GLASS, D. GRIES, A. HALL, M. G. HINCHEY, C. M. HOLLOWAY, D. JACKSON, C. B. JONES, M. J. LUTZ, D. L. PARNAS, J. RUSHBY, J. WING, AND P. ZAVE, "An Invitation to Formal Methods," *IEEE Computer* **29** (April 1996), pp. 16–30. (Chapter 11)

[Sammet, 1978] J. E. SAMMET, "The Early History of COBOL," *Proceedings of the History of Programming Languages Conference,* Los Angeles, 1978, pp. 199–276. (Chapter 14)

[Schach, 1992] S. R. SCHACH, *Software Reuse: Past, Present, and Future,* videotape, 150 mins, US-VHS format. IEEE Computer Society Press, Los Alamitos, CA, November 1992. (Chapter 8)

[Schach, 1994] S. R. SCHACH, "The Economic Impact of Software Reuse on Maintenance," *Journal of Software Maintenance—Research and Practice* **6** (July/August 1994), pp. 185–96. (Chapters 8 and 9)

[Schach, 1996] S. R. SCHACH, "The Cohesion and Coupling of Objects," *Journal of Object-Oriented Programming* **8** (January 1996), pp. 48–50. (Chapter 7)

[Schach, 1997] S. R. SCHACH, *Software Engineering with Java,* Richard D. Irwin, Chicago, 1997. (Chapter 8)

[Schach and Stevens-Guille, 1979] S. R. SCHACH AND P. D. STEVENS-GUILLE, "Two Aspects of Computer-Aided Design," *Transactions of the Royal Society of South Africa* **44** (Part 1, 1979), 123–26. (Chapter 7)

[Schach and Wood, 1986] S. R. SCHACH AND P. T. WOOD, "An Almost Path-Free Very High-Level Interactive Data Manipulation Language for a Microcomputer-Based Database System," *Software—Practice and Experience* **16** (March 1986), pp. 243–68. (Chapter 10)

[Scheffer, Stone, and Rzepka, 1985] P. A. SCHEFFER, A. H. STONE, III, AND W. E. RZEPKA, "A Case Study of SREM," *IEEE Computer* **18** (April 1985), pp. 47–54. (Chapter 11)

[Schmid, 1996] H. A. SCHMID, "Creating Applications from Components: A Manufacturing Framework Design," *IEEE Software* **13** (November/December 1996), pp. 67–75. (Chapter 8)

[Schmidt, 1995] D. C. SCHMIDT, "Using Design Patterns to Develop Reusable Object-Oriented Communications Software," *Communications of the ACM* **38** (October 1995), pp. 65–74. (Chapter 8)

[Scholtz et al., 1993] J. SCHOLTZ, S. CHIDAMBER, R. GLASS, A. GOERNER, M. B. ROSSON, M. STARK, AND I. VESSEY, "Object-Oriented Programming: The Promise and the Reality," *Journal of Systems and Software* **23** (November 1993), pp. 199–204. (Chapter 1)

[Schricker, 2000] D. SCHRICKER, "Cobol for the Next Millennium, *IEEE Software* **17** (March/April 2000), pp. 48–52. (Chapter 8)

[Schwartz and Delisle, 1987] M. D. SCHWARTZ AND N. M. DELISLE, "Specifying a Lift Control System with CSP," *Proceedings of the Fourth International Workshop on Software Specification and Design*, Monterey, CA, April 1987, pp. 21–27. (Chapter 11)

[Sears and Lund, 1997] A. SEARS AND A. M. LUND, "Creating Effective User Interfaces," *IEEE Software* **14** (July/August 1997), pp. 21–25. (Chapter 10)

[SEI, 2000] "Capability Maturity Model Integration (CMMI): Frequently Asked Questions (FAQ)," Version 1.3, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Spring 2000. (Chapter 2)

[Selby, 1989] R. W. SELBY, "Quantitative Studies of Software Reuse," in: *Software Reusability.* Volume 2. *Applications and Experience*, T. J. Biggerstaff and A. J. Perlis (Editors), ACM Press, New York, 1989, pp. 213–33. (Chapter 8)

[Selic, Gullekson, and Ward, 1995] B. SELIC, G. GULLEKSON, AND P. T. WARD, *Real-Time Object-Oriented Modeling*, John Wiley and Sons, New York, 1995. (Chapter 12)

[Shapiro, 1994] F. R. SHAPIRO, "The First Bug," *Byte* **19** (April 1994), p. 308. (Chapter 1)

[Shaw, 1995] M. SHAW, "Comparing Architectural Design Styles," *IEEE Software* **12** (November 1995), pp. 27–41. (Chapter 8)

[Shaw and Garlan, 1996] M. SHAW AND D. GARLAN, *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, Upper Saddle Valley River, NJ, 1996. (Chapter 8)

[Shepperd, 1988a] M. J. SHEPPERD, "An Evaluation of Software Product Metrics," *Information and Software Technology* **30** (No. 3, 1988), pp. 177–88. (Chapter 9)

[Shepperd, 1988b] M. SHEPPERD, "A Critique of Cyclomatic Complexity as a Software Metric," *Software Engineering Journal* **3** (March 1988), pp. 30–36. (Chapter 14)

[Shepperd, 1990] M. SHEPPERD, "Design Metrics: An Empirical Analysis," *Software Engineering Journal* **5** (January 1990), pp. 3–10. (Chapter 13)

[Shepperd, 1996] M. SHEPPERD, *Foundations of Software Measurement*, Prentice Hall, Upper Saddle River, NJ, 1996. (Chapter 5)

[Shepperd and Ince, 1994] M. SHEPPERD AND D. C. INCE, "A Critique of Three Metrics," *Journal of Systems and Software* **26** (September 1994), pp. 197–210. (Chapters 9 and 14)

[Sherer, 1991] S. A. SHERER, "A Cost-Effective Approach to Testing," *IEEE Software* **8** (March 1991), pp. 34–40. (Chapter 15)

[Sherer, Kouchakdjian, and Arnold, 1996] S. W. SHERER, A. KOUCHAKDJIAN, AND P. G. ARNOLD, "Experience Using Cleanroom Software Engineering," *IEEE Software* **13** (May 1996), pp. 69–76. (Chapter 14)

[Shlaer and Mellor, 1992] S. SHLAER AND S. MELLOR, *Object Lifecycles: Modeling the World in States*, Yourdon Press, Englewood Cliffs, NJ, 1992. (Chapters 12 and 13)

[Shneiderman, 1980] B. SHNEIDERMAN, *Software Psychology: Human Factors in Computer and Information Systems*, Winthrop Publishers, Cambridge, MA, 1980. (Chapter 1)

[Shneiderman, 1997] B. SHNEIDERMAN, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 3rd ed., Addison Wesley Longman, Reading, MA, 1997. (Chapter 10)

[Shneiderman and Mayer, 1975] B. SHNEIDERMAN AND R. MAYER, "Towards a Cognitive Model of Programmer Behavior," Technical Report TR-37, Indiana University, Bloomington, 1975. (Chapter 7)

[Siegel, 1998] J. SIEGEL, "OMG Overview: CORBA and the OMA in Enterprise Computing," *Communications of the ACM* **41** (October 1998), pp. 37–43. (Chapter 8)

[Silberschatz and Galvin, 1998] A. SILBERSCHATZ AND P. B. GALVIN, *Operating System Concepts*, 5th ed., Addison-Wesley, Reading, MA, 1998. (Chapters 11 and 13)

[Skazinski, 1994] J. G. SKAZINSKI, "Porting Ada: A Report from the Field," *IEEE Computer* **27** (October 1994), pp. 58–64. (Chapter 8)

[Sneed, 1995] H. M. SNEED, "Planning the Reengineering of Legacy Systems," *IEEE Software* **12** (January 1995), pp. 24–34. (Chapter 16)

[Snyder, 1993] A. SNYDER, "The Essence of Objects: Concepts and Terms," *IEEE Software* **10** (January 1993), pp. 31–42. (Chapter 7)

[Sobell, 1995] M. G. SOBELL, *A Practical Guide to the UNIX System,* 3rd ed., Benjamin/Cummings, Menlo Park, CA, 1995. (Chapter 5)

[Sparks, Benner, and Faris, 1996] S. SPARKS, K. BENNER, AND C. FARIS, "Managing Object-Oriented Framework Reuse," *IEEE Computer* **29** (September 1996), pp. 52–61. (Chapters 8 and 9)

[Spivey, 1990] J. M. SPIVEY, "Specifying a Real-Time Kernel," *IEEE Software* **7** (September 1990), pp. 21–28. (Chapter 11)

[Spivey, 1992] J. M. SPIVEY, *The Z Notation: A Reference Manual,* Prentice Hall, New York, 1992. (Chapters 3 and 11)

[Stankovic, 1997] J. A. STANKOVIC, "Real-Time and Embedded Systems," in: *The Computer Science and Engineering Handbook,* A. B. Tucker, Jr. (Editor-in-Chief), CRC Press, Boca Raton, FL, pp. 1709–24. (Chapter 13)

[Stephenson, 1976] W. E. STEPHENSON, "An Analysis of the Resources Used in Safeguard System Software Development," Bell Laboratories, Draft Paper, August 1976. (Chapter 1)

[Sternbach and Okuda, 1991] R. STERNBACH AND M. OKUDA, *Star Trek: The Next Generation, Technical Manual,* Pocket Books, New York, 1991. (Chapter 2)

[Stevens, Myers, and Constantine, 1974] W. P. STEVENS, G. J. MYERS, AND L. L. CONSTANTINE, "Structured Design," *IBM Systems Journal* **13** (No. 2, 1974), pp. 115–39. (Chapter 7)

[Stocks and Carrington, 1996] P. STOCKS AND D. CARRINGTON, "A Framework for Specification-Based Testing," *IEEE Transactions on Software Engineering* **22** (November 1996), pp. 777–93. (Chapter 14)

[Stolper, 1999] S. A. STOLPER, "Streamlined Design Approach Lands Mars Pathfinder," *IEEE Software* **16** (September/October 1999), pp. 52–62. (Chapter 13)

[Stroustrup, 1991] B. STROUSTRUP, *The C++ Programming Language,* 2nd ed., Addison-Wesley, Reading, MA, 1991. (Chapters 7 and 13)

[Sykes and McGregor, 2000] D. A. SYKES AND J. D. MCGREGOR, *Practical Guide to Testing Object-Oriented Software,* Addison-Wesley, Reading, MA, 2000. (Chapter 6)

[Symons, 1991] C. R. SYMONS, *Software Sizing and Estimating: Mk II FPA,* John Wiley and Sons, Chichester, U.K., 1991. (Chapter 9)

[Szyperski, 1998] C. SZYPERSKI, *Component Software: Beyond Object-Oriented Programming,* Addison-Wesley, Reading, MA, 1998. (Chapter 8)

[Takahashi and Kamayachi, 1985] M. TAKAHASHI AND Y. KAMAYACHI, "An Empirical Study of a Model for Program Error Prediction," *Proceedings of the Eighth International Conference on Software Engineering,* London, 1985, pp. 330–36. (Chapter 14)

[Tanenbaum, 1996] A. S. TANENBAUM, *Computer Networks,* 3rd ed., Prentice Hall, Upper Saddle River, NJ, 1996. (Chapter 8)

[Teichroew and Hershey, 1977] D. TEICHROEW AND E. A. HERSHEY, III, "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering* **SE-3** (January 1977), pp. 41–48. (Chapter 11)

[Teng, Jeong, and Grover, 1998] J. T. C. TENG, S. R. JEONG, AND V. GROVER, "Profiling Successful Reengineering Projects," *Communications of the ACM* **41** (June 1999), pp. 96–102. (Chapter 16)

[Tepfenhart and Cusick, 1997] W. M. TEPFENHART AND J. J. CUSICK, "A Unified Object Topology," *IEEE Software* **14** (January/February 1997), pp. 31-35. (Chapter 8)

[Tevonen, 1996] I. TEVONEN, "Support for Quality-Based Design and Inspection," *IEEE Software* **13** (January 1996), pp. 44–54. (Chapter 6)

[Thayer, 1997] R. H. THAYER, *Software Engineering Project Management,* 2nd ed., IEEE Computer Society Press, Los Alamitos, CA, 1997. (Chapter 9)

[Thayer and Dorfman, 1999] R. H. THAYER AND M. DORFMAN, EDITORS, *Software Requirements Engineering,* rev. 2nd ed., IEEE Computer Society Press, Los Alamitos, CA, 1999. (Chapter 10)

[Thomas, 1989] I. THOMAS, "PCTE Interfaces: Supporting Tools in Software Engineering Environments," *IEEE Software* **6** (November 1989), pp. 15–23. (Chapter 15)

[Tichy, 1985] W. F. TICHY, "RCS—A System for Version Control," *Software—Practice and Experience* **15** (July 1985), pp. 637–54. (Chapters 5, 15, and 16)

[Toft, Coleman, and Ohta, 2000] P. TOFT, D. COLEMAN, AND J. OHTA, "A Cooperative Model for Cross-Divisional Product Development for a Software Product Line," in: *Software Product Lines:*

Experience and Research Directions, P. Donohoe (Editor), Kluwer Academic Publishers, Boston, 2000, pp. 111–32. (Chapter 8)

[Tracz, 1979] W. J. TRACZ, "Computer Programming and the Human Thought Process," *Software—Practice and Experience* **9** (February 1979), pp. 127–37. (Chapter 5)

[Tracz, 1994] W. TRACZ, "Software Reuse Myths Revisited," *Proceedings of the 16th International Conference on Software Engineering*, Sorrento, Italy, May 1994, pp. 271–72. (Chapter 8)

[Tracz, 1995] W. TRACZ, *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison-Wesley, Reading, MA, 1995. (Chapter 8)

[Trammel, Binder, and Snyder, 1992] C. J. TRAMMEL, L. H. BINDER, AND C. E. SNYDER, "The Automated Production Control Documentation System: A Case Study in Cleanroom Software Engineering," *ACM Transactions on Software Engineering and Methodology* **1** (January 1992), pp. 81–94. (Chapter 14)

[Turner, 1994] C. D. TURNER, "State-Based Testing: A New Method for the Testing of Object-Oriented Programs," Ph.D. thesis, Computer Science Division, University of Durham, Durham, U.K., November, 1994. (Chapter 14)

[USNO, 2000] "The 21st Century and the 3rd Millennium—When Will They Begin?" U.S. Naval Observatory, Astronomical Applications Department, aa.usno.navy.mil/AA/faq/docs/millennium.html, February 22, 2000. (Chapter 12)

[Valesky, 1999] T. C. VALESKY, *Enterprise JavaBeans: Developing Component-Based Distributed Applications*, Addison-Wesley, Reading, MA, 1999. (Chapter 8)

[van der Poel and Schach, 1983] K. G. VAN DER POEL AND S. R. SCHACH, "A Software Metric for Cost Estimation and Efficiency Measurement in Data Processing System Development," *Journal of Systems and Software* **3** (September 1983), pp. 187–91. (Chapter 9)

[van Wijngaarden et al., 1975] A. VAN WIJNGAARDEN, B. J. MAILLOUX, J. E. L. PECK, C. H. A. KOSTER, M. SINTZOFF, C. H. LINDSEY, L. G. L. T. MEERTENS, AND R. G. FISKER, "Revised Report on the Algorithmic Language ALGOL 68," *Acta Informatica* **5** (1975), pp. 1–236. (Chapter 2)

[Vlissides, 1998] J. VLISSIDES, *Pattern Hatching: Design Patterns Applied*, Addison-Wesley, Reading, MA, 1998. (Chapter 8)

[Voas, 1999] J. VOAS, "Software Quality's Eight Greatest Myths," *IEEE Software* **16** (September/October 1999), pp. 118–120. (Chapter 6)

[Volta, 1993] L. G. VOLTA, JR., "Does Every Inspection Need a Meeting?" *Proceedings of the First ACM SIGSOFT Symposium on the Foundations of Software Engineering, ACM SIGSOFT Software Engineering Notes* **18** (December 1993), pp. 107–14. (Chapter 6)

[von Mayrhauser and Vana, 1997] A. VON MAYRHAUSER AND A. M. VANA, "Identification of Dynamic Comprehension Processes during Large Scale Maintenance," *IEEE Transactions on Software Engineering* **22** (June 1996), pp. 424–37. (Chapter 16)

[W3C, 2000] "Canonical XML, Version 1.0," W3C Working Draft, World Wide Web Consortium, Massachusetts Institute of Technology Laboratory for Computer Science, Boston MA; Institut National de Recherche en Informatique et en Automatique, France; Keio University, Shonan Fujisawa Campus, Japan, June 1, 2000. (Chapter 8)

[Walker, 1992] I. J. WALKER, "Requirements of an Object-Oriented Design Method," *Software Engineering Journal* **7** (March 1992), pp. 102–13. (Chapter 13)

[Wallis, 1982] P. J. L. WALLIS, *Portable Programming*, John Wiley and Sons, New York, 1982. (Chapter 8)

[Walsh, 1979] T. J. WALSH, "A Software Reliability Study Using a Complexity Measure," *Proceedings of the AFIPS National Computer Conference*, New York, 1979, pp. 761–68. (Chapter 14)

[Ward and Mellor, 1985] P. T. WARD AND S. MELLOR, *Structured Development for Real-Time Systems*, Volumes 1, 2, and 3, Yourdon Press, New York, 1985. (Chapter 13)

[Warnier, 1976] J. D. WARNIER, *Logical Construction of Programs*, Van Nostrand Reinhold, New York, 1976. (Chapters 11 and 13)

[Wasserman, 1996] A. I. WASSERMAN, "Toward a Discipline of Software Engineering," *IEEE Software* **13** (November/December 1996), pp. 23–31. (Chapters 1 and 2)

[Wasserman and Pircher, 1987] A. I. WASSERMAN AND P. A. PIRCHER, "A Graphical, Extensible Integrated Environment for Software Development," Proceedings of the Second ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development

Environments, *ACM SIGPLAN Notices* **22** (January 1987), pp. 131–42. (Chapter 15)

[Webster, 1995] B. F. WEBSTER, *Pitfalls of Object-Oriented Development,* M&T Books, New York, 1995. (Chapter 1)

[Wegner, 1992] P. WEGNER, "Dimensions of Object-Oriented Modeling," *IEEE Computer* **25** (October 1992), pp. 12–20. (Chapter 8)

[Weinberg, 1971] G. M. WEINBERG, *The Psychology of Computer Programming,* Van Nostrand Reinhold, New York, 1971. (Chapters 1 and 4)

[Weinberg, 1992] G. M. WEINBERG, *Quality Software Management: Systems Thinking,* Volume 1, Dorset House, New York, 1992. (Chapter 9)

[Weinberg, 1993] G. M. WEINBERG, *Quality Software Management: First-Order Measurement,* Volume 2, Dorset House, New York, 1993. (Chapter 9)

[Weinberg, 1994] G. M. WEINBERG, *Quality Software Management: Congruent Action,* Volume 3, Dorset House, New York, 1994. (Chapter 9)

[Weinberg, 1997] G. M. WEINBERG, *Quality Software Management: Anticipating Change,* Volume 4, Dorset House, New York, 1997. (Chapter 9)

[Weller, 1993] E. F. WELLER, "Lessons from Three Years of Inspection Data," *IEEE Software* **10** (September 1993), pp. 38–45. (Chapter 6)

[Weller, 1994] E. F. WELLER, "Using Metrics to Manage Software Projects," *IEEE Computer* **27** (September 1994), pp. 27–34 (Chapter 9)

[Wells, 1996] T. D. WELLS, "A Technical Comparison of Borland ObjectWindows 2.0 and Microsoft MFC 2.5," www.it.rit.edu/~tdw/refs/om.htm, February 5, 1996. (Chapter 8)

[Weyuker, 1988a] E. J. WEYUKER, "An Empirical Study of the Complexity of Data Flow Testing," *Proceedings of the Second Workshop on Software Testing, Verification, and Analysis,* Banff, Canada, July 1988, pp. 188–95. (Chapter 14)

[Weyuker, 1988b] E. WEYUKER, "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering* **14** (September 1988), pp. 1357–65. (Chapters 9 and 14)

[Weyuker, 1998] E. J. WEYUKER, "Testing Component-Based Software: A Cautionary Tale," *IEEE Software* **15** (September/October 1998), pp. 54–59. (Chapter 8)

[Wheeler, Brykczynski, and Meeson, 1996] D. A. WHEELER, B. BRYKCZYNSKI, AND R. N. MEESON, JR., *Software Inspection: An Industry Best Practice,* IEEE Computer Society, Los Alamitos, CA, 1996. (Chapter 6)

[Whitgift, 1991] D. WHITGIFT, *Methods and Tools for Software Configuration Management,* John Wiley and Sons, New York, 1991. (Chapter 5)

[Whittaker, 2000] J. A. WHITTAKER, "What Is Software Testing? And Why Is It So Hard?" *IEEE Software* **17** (January/February 2000), pp. 70–79. (Chapter 6)

[Whitten, 1995] N. M. WHITTEN, *Managing Software Development Projects,* 2nd ed., John Wiley and Sons, New York, 1995. (Chapter 9)

[Wilde, Matthews, and Huitt, 1993] N. WILDE, P. MATTHEWS, AND R. HUITT, "Maintaining Object-Oriented Software," *IEEE Software* **10** (January 1993), pp. 75–80. (Chapters 14 and 16)

[Williams, 1996] J. D. WILLIAMS, "Managing Iteration in OO Projects," *IEEE Computer* **29** (September 1996), pp. 39–43. (Chapters 9 and 12)

[Williams, Kessler, Cunningham, and Jeffries, 2000] L. WILLIAMS, R. R. KESSLER, W. CUNNINGHAM, AND R. JEFFRIES, "Strengthening the Case for Pair Programming," *IEEE Software* **17** (July/August 2000), pp. 19–25. (Chapters 3 and 4)

[Wilson, Rosenstein, and Shafer, 1990] D. A. WILSON, L. S. ROSENSTEIN, AND D. SHAFER, *Programming with MacApp,* Addison-Wesley, Reading, MA, 1990. (Chapter 8)

[Wing, 1990] J. WING, "A Specifier's Introduction to Formal Methods," *IEEE Computer* **23** (September 1990), pp. 8–24. (Chapter 11)

[Wirfs-Brock, Wilkerson, and Wiener, 1990] R. WIRFS-BROCK, B. WILKERSON, AND L. WIENER, *Designing Object-Oriented Software,* Prentice Hall, Englewood Cliffs, NJ, 1990. (Chapters 1, 12, and 13)

[Wirth, 1971] N. WIRTH, "Program Development by Stepwise Refinement," *Communications of the ACM* **14** (April 1971), pp. 221–27. (Chapters 5 and 6)

[Wirth, 1975] N. WIRTH, *Algorithms + Data Structures = Programs,* Prentice Hall, Englewood Cliffs, NJ, 1975. (Chapter 5)

[Wohlwend and Rosenbaum, 1993] H. WOHLWEND AND S. ROSENBAUM, "Software Improvements in an International Company," *Proceedings of the 15th International Conference on Software Engineering,* Baltimore, May 1993, pp. 212–20. (Chapter 2)

[Woodcock, 1989] J. WOODCOCK, "Calculating Properties of Z Specifications," *ACM SIGSOFT Software Engineering Notes* **14** (July 1989), pp. 43–54. (Chapter 11)

[Woodward, Hedley, and Hennell, 1980] M. R. WOODWARD, D. HEDLEY, AND M. A. HENNELL, "Experience with Path Analysis and Testing of Programs," *IEEE Transactions on Software Engineering* **SE-6** (May 1980), pp. 278–86. (Chapter 14)

[Yamaura, 1998] T. YAMAURA, "How to Design Practical Test Cases," *IEEE Software* **15** (November/December 1998), pp. 30–36. (Chapter 14)

[Yourdon, 1989] E. YOURDON, *Modern Structured Analysis,* Yourdon Press, Englewood Cliffs, NJ, 1989. (Chapters 5 and 15)

[Yourdon, 1996] E. YOURDON, *Rise and Resurrection of the American Programmer,* Yourdon Press, Upper Saddle River, NJ, 1996. (Chapter 1)

[Yourdon and Constantine, 1979] E. YOURDON AND L. L. CONSTANTINE, *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design,* Prentice Hall, Englewood Cliffs, NJ, 1979. (Chapters 7, 11, and 13)

[Zage and Zage, 1993] W. M. ZAGE AND D. M. ZAGE, "Evaluating Design Metrics on Large-Scale Software," *IEEE Software* **10** (July 1993), pp. 75–81. (Chapter 13)

[Zelkowitz, Shaw, and Gannon, 1979] M. V. ZELKOWITZ, A. C. SHAW, AND J. D. GANNON, *Principles of Software Engineering and Design,* Prentice Hall, Englewood Cliffs, NJ, 1979. (Chapter 1)

[Zvegintzov, 1998] N. ZVEGINTZOV, "Frequently Begged Questions and How to Answer Them," *IEEE Software* **15** (January/February 1998), pp. 93–96. (Chapter 1)

색 인

ㄱ

ㄴ

ㄷ

ㄹ

ㅁ

ㅂ

ㅅ

ㅈ

ㅊ

ㅋ

ㅌ

ㅍ

ㅎ

ㅆ

ㅇ